

Figura.1 UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN SISTEMAS COMPUTACIONALES



**TRABAJO DE GRADO PREVIA LA OBTENCIÓN DEL TÍTULO DE
INGENIERÍA EN SISTEMAS COMPUTACIONALES**

TEMA:

“La Programación Extrema aplicada al desarrollo del Sistema Informático para la Gestión de Fondos de la Asociación de Profesores de la FICA utilizando MVC”

Autor:

GUZMÁN ANGULO LORENA MAGALI

Director: Ing. Miguel Orquera

Ibarra, 2012

AGRADECIMIENTO

Agradezco primeramente a Dios por concederme la vida, a mis padres Germán y Yolanda, mis hermanos Andrés y Cristina, mis tíos Luis y Anita, quienes me ayudaron y apoyaron incondicionalmente para la culminación de este trabajo de tesis.

Además quiero agradecer a mi tutor Ing. Miguel Orquera por guiarme en el desarrollo del Sistema SIGFAP, quien con sus conocimientos hizo posible que haya finalizado mi trabajo.

A si mismo agradezco a mi familia y amigos por estar junto a mí.

DEDICATORIA

Dedico este trabajo de tesis a mis Padres por su dedicación por muchos años, apoyo incondicional, porque con su ayuda he podido realizarme como profesional.



UNIVERSIDAD TÉCNICA DEL NORTE
BIBLIOTECA UNIVERSITARIA
AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR
DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

La universidad Técnica del Norte dentro del proyecto Repositorio Digital Institucional determina la necesidad de disponer de textos completos en formato digital con la finalidad de apoyar los procesos de investigación, docencia y extensión de la Universidad.

Por medio del presente documento dejo sentada mi voluntad de participar en este proyecto para lo cual pongo a disposición la siguiente información.

DATOS DE CONTACTO	
CÉDULA DE IDENTIDAD	0401308069
APELLIDOS Y NOMBRES	GUZMÁN ANGULO LORENA MAGALI
DIRECCIÓN	Elías Almeida 3-17 y José Nicolás Vacas
EMAIL	guzmanloren@gmail.com
TELÉFONO MÓVIL	089944067

DATOS DE LA OBRA	
TITULO	LA PROGRAMACIÓN EXTREMA APLICADA AL DESARROLLO DEL SISTEMA INFORMÁTICO PARA LA GESTIÓN DE FONDOS DE LA ASOCIACIÓN DE PROFESORES DE LA FICA UTILIZANDO MVC.
AUTOR	GUZMÁN ANGULO LORENA MAGALI
FECHA	Enero de 2012
PROGRAMA	PREGRADO
TÍTULO POR EL QUE OPTA	INGENIERÍA EN SISTEMAS COMPUTACIONALES
DIRECTOR	ING. MIGUEL ORQUERA

2. AUTORIZACIÓN DE USO A FAVOR DE LA UNIVERSIDAD

Yo, Lorena Magali Guzmán Angulo, con cédula de identidad Nro. 040130806-9, en calidad de autor y titular de los derechos patrimoniales de la obra o trabajo de grado descrito anteriormente, hago entrega del ejemplar respectivo en forma digital y autorizo a la Universidad Técnica del Norte, la publicación de la obra en el Repositorio Digital Institucional y uso del archivo digital en le biblioteca de la universidad con fines académicos. Para ampliar la disponibilidad del material y como apoyo a la educación, investigación y extensión, en concordancia con la Ley de Educación Superior Artículo 143.



UNIVERSIDAD TÉCNICA DEL NORTE
CESIÓN DE DERECHO DE AUTOR DEL TRABAJO DE
GRADO A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL
NORTE

Yo, Lorena Magali Guzmán Angulo, con cédula de identidad Nro. 040130806-9, manifiesto mi voluntad de ceder a la Universidad Técnica del Norte los derechos patrimoniales consagrados en la Ley de Propiedad Intelectual del Ecuador, artículo 4. 5 y 6 en calidad de autor del trabajo de grado denominado: **“LA PROGRAMACIÓN EXTREMA APLICADA AL DESARROLLO DEL SISTEMA INFORMÁTICO DE GESTIÓN DE FONDOS DE LA ASOCIACIÓN DE PROFESORES DE LA FICA UTILIZANDO MVC”**, que ha sido desarrollado para optar por el título de: Ingeniero en Sistemas Computacionales, quedando la Universidad Técnica del Norte facultada para ejercer plenamente los derechos cedidos anteriormente.

En mi condición de autor de reserva los derechos morales de la obra antes citada.

En concordancia suscribo este documento en el momento en el que hago la entrega del trabajo final en formato impreso y digital a la biblioteca de la Universidad Técnica del Norte.

Firma

Nombre: Lorena Magali Guzmán Angulo

Cédula: 040130806-9

Ibarra, Enero de 2012

CERTIFICACIÓN

Una vez revisado el proyecto: **“LA PROGRAMACIÓN EXTREMA APLICADA AL DESARROLLO DEL SISTEMA INFORMÁTICO DE GESTIÓN DE FONDOS DE LA ASOCIACIÓN DE PROFESORES DE LA FICA UTILIZANDO MVC”**, realizado por la Srta. Lorena Magali Guzmán Angulo, con cédula de identidad Nro. 040130806-9, certifico que el mencionado proyecto fue realizado en su totalidad por la Srta. Lorena Guzmán.

Ing. Miguel Orquera
DIRECTOR DE TESIS

Asociación de Profesores de la FICA

CARTA DE ACEPTACIÓN

Ibarra, 8 de Diciembre de 2011

Por este medio la Asociación de Profesores de la Facultad de Ingeniería en Ciencias Aplicadas, acepta y da por concluidos los trabajos de desarrollo de la tesis, “La Programación Extrema aplicada al desarrollo del sistema informático de gestión de fondos de la asociación de profesores de la FICA utilizando MVC”, realizado por la Srta. Lorena Magali Guzmán Angulo con cédula de identidad 040130806-9, después de haber realizado las pruebas necesarias y suficientes con las que ha demostrado que cumple satisfactoriamente con los requerimientos expuestos en las reuniones sostenidas con la Directiva de la Asociación de Profesores.

Durante las pruebas realizadas, se ha validado que han sido atendidos los requerimientos solicitados y funcionan de manera satisfactoria, además de otras mejoras funcionales incluidas en el aplicativo.

Ing. Miguel Orquera

TESORERO DE LA ASOCIACIÓN DE PROFESORES DE LA FICA

Resumen

Los métodos ligeros o ágiles son otra opción para el desarrollo, es un proceso Incremental, cooperativo, sencillo y finalmente adaptativo. Es una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación. Los métodos ágiles resaltan las comunicaciones cara a cara en vez de la documentación. La mayoría de los equipos ágiles están localizados en una oficina abierta, llamadas "plataformas de lanzamiento" (bullpen en inglés). En las metodologías ágiles se intenta ser lo más flexible posible, que el cliente pueda cambiar los requisitos cuando quiera y que el código funcione bien. Existen muchos métodos de desarrollo ágil; la mayoría minimiza riesgos desarrollando software en cortos lapsos de tiempo, conocidos como iteraciones que van de una a cuatro semanas.

Una de las metodologías de desarrollo ágil es XP Xtreme Programming la cual hace énfasis en los siguientes aspectos: satisfacción del cliente y trabajo en equipo. La metodología propone que un proyecto debe adaptarse a los cambios de requisitos en cualquier punto de su ciclo de vida. Está basada en los siguientes principios: Simplicidad, Comunicación, Retroalimentación (feedback) y Coraje. Además el ciclo de vida de la Programación Extrema está conformada por los siguientes puntos: Planificación, Diseño, Codificación y Pruebas. Estas fases no necesariamente deben realizarse en ese orden si no que se deben realizar de acuerdo a las actividades que se realicen. Para el desarrollo del Sistema de Gestión de Fondos de la Asociación de Profesores de la FICA (SIGFAP) se aplicará la metodología XP, permitiendo que el proyecto se adapte a los requerimientos del usuario en cualquier punto de su ciclo de vida.

Una alternativa para el desarrollo del sistema SIGFAP es utilizar una arquitectura MVC conocida como Modelo, Controlador, Vista, que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

Abstract

Light or agile methods are another option for development is an incremental process, cooperative, and finally simple adaptive. It is an alternative to the processes of traditional software development, characterized by being rigid and directed by the documentation. Agile methods emphasize face-to-face communication rather than documentation. Most agile teams are located in an open office, called "bullpen". In agile methodologies try to be as flexible as possible, so the customer can change the requirements at any time and that the code works well.

There are many agile development methods, most developing software minimizes risk over short periods of time, known as iterations ranging from one to four weeks. One of the agile development methodologies Xtreme Programming XP which emphasizes the following areas: customer satisfaction and work team. La methodology proposes that a project must adapt to changing requirements at any point in its life cycle. It is based on the following principles: Simplicity, Communication, Feedback (feedback) and Courage. In addition, the life cycle of Extreme Programming consists of the following: Planning, Design, Coding and Testing. These phases need not be in that order if not to be performed according to the activities undertaken.

For the development of the "Sistema Informático de Gestión de Fondos de la Asociación de la FICA (SIGFAP) applies the XP methodology, allowing the project meets the requirements of the user at any point in its life cycle.

In addition to developing an alternative system is used SIGFAP MVC architecture known as Model, Controller, View, which separates the application data, user interface and control logic into three distinct components.

Índice de Contenido

1. Capítulo I	1
1.1. METODOLOGÍAS ÁGILES DE DESARROLLO DE SOFTWARE	1
1.1.1. INTRODUCCION	1
1.1.2. DEFINICION.....	2
1.1.3. CARACTERISTICAS DE LAS METODOLOGÍAS ÁGILES.....	3
1.1.4. PRINCIPIOS DE LAS METODOLOGÍAS ÁGILES	4
1.1.5. TIPOS DE METODOLOGIAS ÁGILES	7
1.1.6. XP EXTREME PROGRAMMING.....	9
1.1.6.1.Elementos de la metodología XP.	10
1.1.6.1.1. Historias del usuario.....	10
1.1.6.2.Roles XP.....	10
1.1.6.3.Proceso XP.....	11
1.1.6.4.Prácticas	11
1.1.6.5.Principio de XP.	13
1.1.6.6.Actividades de XP.	15
1.1.6.6.1. Codificar.	15
1.1.6.6.2. Probar.....	15
1.1.6.6.3. Escuchar.	16
1.1.6.6.4. Diseñar	16
1.1.7. SCRUM.....	17
1.1.7.1.Características	17
1.1.7.2.PRÁCTICAS DE SCRUM	19
1.1.7.3.CICLO DE VIDA DE SCRUM.....	20
1.1.8. ASD – ADAPTIVE SOFTWARE DEVELOPMENT	22
1.1.8.1.CARACTERÍSTICAS DE ASD.....	23

1.1.8.2.	ASPECTOS CLAVES DE ASD	24
1.1.8.3.	CICLO DE VIDA DE ASD	24
1.1.8.3.1.	Especlar	24
1.1.8.3.2.	Colaborar.....	25
1.1.8.3.3.	Aprender	25
1.1.9.	AUP.....	26
1.1.9.1.	FASES DE AUP.....	27
1.1.9.2.	DISCIPLINAS DE AUP.....	28
1.1.9.3.	FILOSOFÍAS DE AUP.....	29
1.1.9.4.	RELEASE.....	30
1.1.10.	LSD LEAN PRODUCT DEVELOPMENT.....	30
1.1.10.1	PRINCIPIOS	31
1.2.	METODOLOGÍAS TRADICIONALES DE DESARROLLO DE SOFTWARE.....	35
1.2.1.	EJEMPLOS DE METODOLOGÍAS TRADICIONALES.....	37
1.2.1.1.	BASADAS EN CICLOS DE VIDA.....	37
1.2.1.1.1.	EN CASCADA.....	37
1.2.1.1.2.	PROTOTIPADO RÁPIDO	37
1.2.1.1.3.	EVOLUTIVO.....	38
1.2.1.1.4.	INCREMENTAL.....	38
1.2.1.1.5.	EN ESPIRAL	38
1.2.1.2.	METODOLOGÍAS ORIENTADAS A PROCESOS	38
1.2.1.2.1.	RationalUnifiedProcess (RUP)	38
1.2.1.2.2.	Microsoft Solution Framework (MSF).....	39
1.3.	COMPARACIONES DE LAS METODOLOGÍAS TRADICIONALES Y METODOLOGÍAS ÁGILES.....	39
1.4.	VENTAJAS DEL USO DE METODOLOGÍAS ÁGILES.....	42
1.5.	CONCLUSIONES	44
2.	Capítulo III	45
2.1.	METODOLOGÍA DE PROGRAMACIÓN EXTREMA XP.....	45

2.1.1.	POR QUE UTILIZAR XP?.....	45
2.1.2.	DEFINICIÓN DE PROGRAMACIÓN EXTREMA O EXTREME PROGRAMMING XP	46
2.1.3.	CONTEXTO XP.....	47
2.1.4.	AXIOMAS DE XP	47
2.1.5.	DERECHOS DE CLIENTES Y DESARROLADORES EN XP	48
2.1.5.1.	Derechos del Cliente	48
2.1.5.2.	Derechos del Desarrollador	48
2.1.6.	RESPONSABILIDADES DE CLIENTES Y DESARROLADORES EN XP	49
2.1.6.1.	Responsabilidades de los Clientes	49
2.1.6.2.	Responsabilidades de los Desarrolladores.....	49
2.1.7.	ALGUNOS ASPECTOS CULTURALES DE XP	49
2.1.8.	CARACTERÍSTICAS DE XP	50
2.1.9.	ARTEFACTOS DE LA METODOLOGÍA XP	51
2.1.9.1.	LAS HISTORIAS DE USUARIO O USER STORIES	51
2.1.9.1.1.	DETALLANDO UNA HISTORIA DE USUARIO.....	56
2.1.9.1.2.	DESCRIPCIÓN ESCRITA.....	57
2.1.9.1.3.	LAS HISTORIAS DEL USUARIO Y PLANEAMIENTO.....	58
2.1.9.1.4.	DIVIDIR UNA USER STORY ÉPICA.....	60
2.1.9.1.5.	USER STORIES EFECTIVAS.....	62
2.1.9.2.	TAREAS DE INGENIERÍA.....	63
2.1.9.3.	PRUEBAS DE ACEPTACIÓN	64
2.1.9.4.	TARJETAS CRC.....	70
2.1.9.4.1.	Cómo crear modelos CRC?.....	72
2.1.10.	ROLES XP	73
2.1.11.	PROCESO XP:	75
2.1.12.	CICLO DE VIDA DE XP.....	75
2.1.12.1.	Exploración:.....	76
2.1.12.2.	Planeamiento:	77

2.1.12.3.Producción:	77
2.1.12.4.Mantenimiento:	78
2.1.12.5.Muerte:	78
2.1.13. VALORES DE LA PROGRAMACIÓN EXTREMA	78
2.1.13.1.SIMPLICIDAD	79
2.1.13.2.COMUNICACIÓN.....	79
2.1.13.3.REALIMENTACIÓN(FEEDBACK):.....	80
2.1.13.4.CORAJE O VALENTÍA.....	81
2.1.13.5.RESPETO	81
2.1.14. PRINCIPIOS BÁSICOS DE LA METODOLOGÍA XP	82
2.1.15. BUENAS PRÁCTICAPARA LA APLICACIÓN EXITOSA DE LA METODOLOGÍA XP	82
2.1.16. ACTIVIDADES DE LA METODOLOGÍA XP	96
2.1.16.1.CODIFICAR	96
2.1.16.2.PROBAR	96
2.1.16.3.ESCUCHAR	97
2.1.16.4.DISEÑAR	97
2.1.17. FASES DE LA METODOLOGÍA XP.....	98
2.1.17.1.PLANIFICACIÓN	98
2.1.17.2.DISEÑO	100
2.1.17.3.DESARROLLO	101
2.1.17.4.PRUEBAS.....	102
2.1.18. VENTAJAS DE UTILIZAR LA METODOLOGÍA XP.....	103
2.1.19. LIMITACIONES	105
3. CAPITULO III.....	107
3.1. ARQUITECTURA MVC	107
3.1.1. DEFINICIÓN.....	107
3.1.2. CARACTERÍSTICAS DE MVC.....	107
3.1.2.1.EL MODELO	108
3.1.2.2.LA VISTA	108

3.1.2.3.EL CONTROLADOR.....	108
3.1.3. CICLO DE VIDA MVC	109
3.1.4. FRAMEWORKS MVC	110
3.1.5. CONSIDERACIONES.....	112
3.2. TECNOLOGÍAS A USAR.....	112
3.2.1. BASE DE DATOS ORACLE.	112
3.2.1.1.ELEMENTOS DEL SERVIDOR ORACLE.....	115
3.2.1.2.INSTANCIA ORACLE	115
3.2.1.3.HERRAMIENTAS ORACLE	116
3.2.1.4.VERSIONES	117
3.2.2. JDeveloper	118
3.2.2.1.VERSIONES DE JDEVELOPER.....	118
3.2.2.2.CICLO DE VIDA DE JDEVELOPER	119
3.2.3. OC4J.....	120
3.2.3.1.Pre-requisitos	120
3.2.4. Modelo Vista Controlador en JSF	121
3.2.4.1.MODELO	121
3.2.4.2.VISTA	122
3.2.4.3.CONTROLADOR	122
3.3. ARQUITECTURA DEL SISTEMA DE GESTIÓN DE AHORRO PARA LA ASOCIACIÓN DE PROFESORES DE FICA.	123
3.3.1. CICLO DE VIDA MVC PARA SIGFAP	123
4. CAPITULO IV	125
4.1. APLICACIÓN DE LA METODOLOGÍA XP AL PROYECTO	125
4.1.1. PLANIFICACIÓN.....	125
4.1.1.1.HISTORIAS DE USUARIO.	125
4.1.1.1.1. Descripción de las historias de usuario.....	127
4.1.1.2.ACTIVIDADES	127

4.1.1.3.PLAN DE ENTREGAS.....	130
4.1.1.4.TRABAJO EN PAREJAS.....	132
4.1.1.5.PROPIEDAD COLECTIVA DEL CÓDIGO.	132
4.1.2. DISEÑO	133
4.1.2.1.METÁFORA.....	133
4.1.2.2.CRC. Tarjetas de Responsabilidad Colaboración.....	133
4.1.2.2.1. Tarjeta CRC Libro Diario	134
4.1.2.2.2. Tarjeta CRC Detalle Libro Diario.....	135
4.1.3. DESARROLLO	136
4.1.3.1.BASE DE DATOS.....	136
4.1.3.1.1. Diagrama E-R.....	136
4.1.3.2.Prototipos.....	137
4.1.3.2.1. Ventana de Libro Diario en el Release 1.	137
4.1.3.2.2. Ventana de Ingreso de Datos de Libro Diario en el último Release.....	137
4.1.4. PRUEBAS.....	138
4.1.4.1.PRUEBA DE ACEPTACIÓN.	138
5. CAPITULO V	145
5.1. CONCLUSIONES	145
5.2. RECOMENDACIONES	146
5.3. BIBLIOGRAFÍA.....	147

Índice de Figuras.

Figura.1 Convergencias y Divergencias de las principales metodologías ágiles	8
Figura.2 Esquema de trabajo de XP	13
Figura.3 Flujo del proceso de SCRUM	19
Figura.4 Ciclo de vida de Scrum	21
Figura.5 El ciclo de vida adaptativo. Tomada de [Highsmith, 2000].	26
Figura.6 Disciplinas.....	29
Figura.7 Tipos de release.....	30
Figura.8 Principios	35
Figura.9 Metodologías orientadas a procesos	39
Figura.10 Enfoque de las metodologías	41
Figura.11 Historia de Usuario.....	53
Figura.12 Historia de Usuario durante el análisis	56
Figura.13 Historia de Usuario. Descripción escrita.	58
Figura.14 Historia de Usuario. Planeamiento Programado.	59
Figura.15 Tareas de Ingeniería	64
Figura.16 Plantilla de prueba de aceptación.....	65
Figura.17 Pruebas de Aceptación.....	69
Figura.18 Tarjeta CRC.	70
Figura.19 Ejemplo de Tarjeta CRC.....	71
Figura.20 Proceso de XP	75
Figura.21 Ciclo de vida de XP	76
Figura.22 Prácticas de XP	95
Figura.23 Iteraciones de XP	98

Figura.24 Propiedad colectiva del código	99
Figura.25 Fases de XP	103
Figura.26 Patron MVC : http://feeds.feedburner.com/librosweb_es	109
Figura.27 Ciclo de Vida MVC	110
Figura.28 Instancia de Oracle	116
Figura.29 Acceso a la Base de Datos	116
Figura.30 Ciclo de vida de JDeveloper.....	119
Figura.31 MVC en JSF	121
Figura.32 Solicitud de usuario en la Vista- MVC	123
Figura.33 Gestión del Controlador.....	124
Figura.34 Gestión del Modelo.....	124
Figura.35 Información redirigida del Modelo al Controlador.	124
Figura.36 Datos transformados en la Vista	124
Figura.37 Historia de Usuario Registro de Asientos Contables.....	127
Figura.38 Actividad 1 para la Historia de Usuario 1	127
Figura.39 Actividad 2 para la Historia de Usuario 1	128
Figura.40 Actividad 3 para la Historia de Usuario 1	128
Figura.41 Actividad 4 para la Historia de Usuario 1	129
Figura.42 Actividad 5 para la Historia de Usuario 1	129
Figura.43 Actividad 6 para la Historia de Usuario 1	130
Figura.44 Fechas de entregas para cada una de las historias de Usuario.....	132
Figura.45 Tarjeta CRC para el Libro Diario	134
Figura.46 Tarjeta CRC para el Detalle de Libro Diario	135
Figura.47 Diagrama E-R de SIGFAP.....	136
Figura.48 Ventana de Ingreso de datos de un Asiento contable en primer release.....	137
Figura.49 Ventana de Ingreso de datos de un Asiento contable en último release	137

Índice de Tablas.

Tabla. 1 Diferencias entre Metodologías ágiles y metodologías tradicionales.	40
Tabla. 2 Axiomas de XP	47
Tabla. 3 Pasos para escribir una prueba de aceptación.....	66
Tabla. 4 Frameworks MVC	112
Tabla. 5 Prioridad de las Historias de usuario.....	131
Tabla. 6 Historias de usuario definidas en una fecha posterior.	131

Índice de Contenido de Anexos

Accesos al Sistema como Administrador	149
Plan de Cuentas.....	150
Asientos Contables.....	152
Préstamos.....	156
Pagos	160
Reportes	166
Cierre de periodos contables	171
Historiales.....	172
Actualización de datos	173
Acceso al Sistema como Operador.....	173
Periodo contable	173
Interés	175
Tipo de cuenta.....	176
Parámetros.....	177
Usuarios.....	178
Acceso al sistema como Auditor	180
Parámetros.....	180
Datos	181
Acceso al sistema como Socio.....	182
Reportes	182
Ganancia.....	184
MANUAL DE INSTALACIÓN	186
Oracle Application Server Standalone - OC4J.....	186
Instalación de Oracle 10g.....	188

Índice de figuras de Anexos.

Fig.1	archivo startoc4j.bat	186
Fig.2	archivo startoc4j.bat - 2.....	186
Fig.3	Ejecución archivo startoc4j.bat	186
Fig.4	Acceso a navegador web	187
Fig.5	Autenticación de usuario oc4j.....	187
Fig.6	Administración oc4j.....	187
Fig.7	Archivos de instalación de Oracle	188
Fig.8	Instalación de Base de datos Oracle.	189
Fig.9	Proceso de Instalación	189
Fig.10	Comprobación de requisitos	190
Fig.11	Resumen de Datos de Instalación	190
Fig.12	Progreso de la instalación.	190
Fig.13	Creación de Base de Datos en curso.	191
Fig.14	Gestión de Contraseñas.	191
Fig.15	Selección de Cuentas Bloqueadas y gestión de contraseñas.....	192
Fig.16	Finalización de Instalación	192
Fig.17	Mensaje de salida de Instalación.	192
Fig.18	Accediendo a Oracle a través del navegador	193
Fig.19	Conexión a la Base de Datos.	193
Fig.20	Acuerdo de Licencia.	194
Fig.21	Resumen de la Base de Datos.	194
Fig.22	Tablespaces de la Base de Datos.....	195
Fig.23	Creación del Tablespace sigfap	195

Fig.24	Usuarios de la Base de Datos	196
Fig.25	Usuario sigfap.....	196
Fig.26	Opciones de Mantenimiento	197

1. Capítulo I

1.1. METODOLOGÍAS ÁGILES DE DESARROLLO DE SOFTWARE

1.1.1. INTRODUCCIÓN

Según el Instituto Nacional de Tecnologías de la Comunicación de España, el software juega un papel significativo en la vida de las personas. Desde hace muchos años el principal objetivo en el desarrollo de software ha sido diseñarlo e implementarlo en menos tiempo y que su coste sea menor.

Existen varias metodologías para desarrollar software. Las metodologías que se han utilizado se caracterizan por ser rígidas y su enfoque va dirigido a la documentación, es así que se centran más en el control del proceso, estableciendo rigurosamente las actividades, herramientas y notaciones. A estos métodos se les suele conocer como métodos tradicionales o pesados.

La aplicación de estas metodologías no resultan ser las más adecuadas para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad.

Es por eso que en el año 2001 un grupo de expertos en software se reunió en Utah-EEUU y formaron el término ágil aplicado al desarrollo de software, en esta reunión se planteó los objetivos y principios que debería tener el desarrollo de software de forma que se realice

rápidamente y este de acuerdo a los cambios que tenga el proyecto. Estas nuevas técnicas fueron conocidas como metodologías ágiles.

Las metodologías ágiles se basan en dos aspectos fundamentales, retrasar las decisiones y la planificación adaptativa al cambio de requisitos, y se enfocan en la gente y los resultados.

Existen muchos métodos de desarrollo ágil; la mayoría minimiza riesgos desarrollando software en cortos lapsos de tiempo, llamado una iteración, la cual debe durar de una a cuatro semanas.

Una iteración no debe agregar demasiada funcionalidad para justificar el lanzamiento del producto al mercado, pero la meta es tener un demo (sin errores) al final de cada iteración. Al final de cada iteración el equipo vuelve a evaluar las prioridades del proyecto.

1.1.2. DEFINICIÓN¹

Se entiende como desarrollo ágil de software al desarrollo de software basado en procesos ágiles conocidos anteriormente como metodologías livianas. Esta metodología busca un justo medio entre ningún proceso y demasiado proceso, proporcionando el suficiente para que el esfuerzo valga la pena.

El término "ágil" es una referencia a la capacidad de adaptarse a los cambios de contexto y a los cambios de especificaciones que ocurren durante el proceso de desarrollo.

Una metodología Ágil es una metodología efectiva para modelar y documentar un proyecto de software, es una colección de valores principios y prácticas para modelar software que pueden ser aplicadas de manera simple y ligera.

¹http://es.wikipedia.org/wiki/Desarrollo_de_software

Las metodologías ágiles promueven generalmente un proceso de gestión de proyectos que fomenta el trabajo en equipo, la organización y responsabilidad propia, un conjunto de mejores prácticas de ingeniería que permiten la entrega rápida de software de alta calidad, y un enfoque de negocio que alinea el desarrollo con las necesidades del cliente y los objetivos de la compañía.

1.1.3. CARACTERÍSTICAS DE LAS METODOLOGÍAS ÁGILES.

Para definir las características que tienen las metodologías ágiles de desarrollo de software se creó el documento de la filosofía ágil.

En la reunión efectuada en el 2001 se creó The Agile Alliance, la cual es una organización sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos.

En la reunión se encontraban 17 expertos dentro de la industria del software, entre ellos creadores e impulsores de metodologías de software, propusieron una alternativa al desarrollo de software diferente a las metodologías tradicionales, el punto de partida fue el Manifiesto Ágil, un documento que resume la filosofía ágil.

Según el Manifiesto se valora:²

- *Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas.* La gente es el principal factor de éxito de un proyecto software. Es más importante construir un buen equipo que construir el entorno.

²<http://www.cenitec.com.mx/Manifiesto.pdf>

- ***Desarrollar software que funciona más que conseguir una buena documentación.***

La regla a seguir es no producir documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante. Estos documentos deben ser cortos y centrarse en lo fundamental.

- ***La colaboración con el cliente más que la negociación de un contrato.*** Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.

- ***Responder a los cambios más que seguir estrictamente un plan.*** La habilidad de responder a los cambios que puedan surgir a lo largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta.

Los valores descritos dan lugar a los principios del manifiesto los cuales son características de un proceso ágil de un tradicional.

Tenemos doce principios, en los cuales los dos primeros principios son generales y resumen gran parte del espíritu ágil. Los demás se relacionan con el proceso a seguir y con el equipo de desarrollo, en cuanto: metas a seguir y organización del mismo.

1.1.4. PRINCIPIOS DE LAS METODOLOGÍAS ÁGILES ³

A continuación se describen los siguientes principios:

1. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.

³<http://seccperu.org/files/Metodologias%20Agiles.pdf>

2. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
3. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.
4. La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.
5. Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.
6. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.
7. El software que funciona es la medida principal de progreso.
8. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.
9. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
10. La simplicidad es esencial.
11. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.
12. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.

Tomando en cuenta los principios descritos se dice que:

Las metodologías ágiles intentan ser lo más flexible posible, tanto así que el cliente pueda cambiar los requisitos cuando quiera y que el código funcione bien.

Las metodologías ágiles resaltan la comunicación cara a cara en lugar de la documentación. La mayoría de los equipos ágiles se encuentran en una oficina abierta, a veces llamadas "plataformas de lanzamiento" (bullpen en inglés). El equipo de trabajo debe incluir: revisores, escritores de documentación y ayuda, diseñadores de iteración, directores de proyecto y clientes o su representante. El representante del cliente debe ser designado por los involucrados en el negocio. Los miembros de los equipos normalmente toman responsabilidades de tareas que consigan la funcionalidad de una iteración. Deciden ellos mismos cómo realizarán las tareas durante una iteración.

Al final de cada iteración, las personas involucradas en el negocio y el cliente o su representante, revisan el progreso y reevalúan las prioridades para optimizar el retorno de la inversión y asegurando la alineación con las necesidades del cliente y los objetivos de la compañía.

La metodología ágil al preferir la comunicación cara a cara, produce menos documentación escrita. Entonces la documentación de las metodologías ágiles debe ser tan simple como sea posible y de acuerdo a la audiencia a la que vaya dirigida, no es un modelado tan prescriptivo, no da recetas de diseño, no crea documentación innecesaria, se puede usar cualquier tipo de diseño o documentación que nos ayude, puede haber procesos que sea difícil capturarlos con los diagramas conocidos, pero si otros.

Para las metodologías ágiles, la documentación es para los desarrolladores, son ellos quienes la necesitan durante su trabajo diario, tanto en periodos de desarrollo como de mantenimiento.

La metodología ágil promueve iteraciones a lo largo del ciclo de vida del proyecto; conocidas como timeboxes, se realiza de forma colaborativa mediante la organización de los equipos que producen software de alta calidad con un coste efectivo y en el tiempo

apropiado que cumple con las necesidades cambiantes de las personas involucradas en el negocio.

Las iteraciones son lapsos de tiempo que van de una a cuatros semanas. Cada iteración del ciclo de vida incluye:

- ✓ Planificación,
- ✓ Análisis de requerimientos,
- ✓ Diseño,
- ✓ Codificación,
- ✓ Revisión y
- ✓ Documentación.

Al final de cada iteración el equipo vuelve a evaluar las prioridades del proyecto. Estas iteraciones permiten que el proyecto se adapte a los cambios rápidamente.

En cada iteración no se debe añadir tanta funcionalidad para que se justifique el lanzamiento de una versión disponible; con errores mínimos.

1.1.5. TIPOS DE METODOLOGÍAS ÁGILES

Existen varias metodologías ágiles, En la Figura.1 se muestran las divergencias y convergencias de algunas de ellas.

Metodología	Acrónimo	Creación	Tipo de modelo	Característica
Adaptive Software Development	ASD	Highsmith 2000	Prácticas + ciclo de vida	Inspirado en sistemas adaptativos complejos
Agile Modeling	AM	Ambler 2002	Metodología basada en la práctica	Suministra modelado ágil a otros métodos
Cristal Methods	CM	Cockburn 1998	Familia de metodologías	Metodología ágil con énfasis en modelo de ciclos
Agile RUP	dX	Booch, Martin, Newkirk 1998	Framework/Disciplina	XP dado vuelta con artefactos RUP
Dynamic Solutions Delivery Model	DSDM	Stapleton 1997	Framework/modelo de ciclo de vida	Creado por 16 expertos en RAD
Evolutionary Project Management	EVO	Gilb 1976	Framework adaptativo	Primer método ágil existente
eXtreme Programming	XP	Beck 1999	Disciplina en prácticas de ingeniería	Método ágil radical
Feature-Driven Development	FDD	De Luca & Coad 1998 Palmer & Felsing 2002	Metodología	Método ágil de diseño y construcción
Lean Development	LD	Charette 2001, Mary y Tom Poppendieck	Forma de pensar - modelo logístico	Metodología basada en procesos productivos
Rapid Development	RAD	McConnell 1996	Survey de técnicas y modelos	Selección de <i>best practices</i> , no método
Microsoft Solutions Framework	MSF	Microsoft 1994	Lineamientos, disciplinas, prácticas	Framework de desarrollo de soluciones
Scrum	Scrum	Sutherland 1994 Schwaber 1995	Proceso - framework de management	Complemento de otros métodos, ágiles o no

Figura.2 Convergencias y Divergencias de las principales metodologías ágiles⁴

Ahora bien, las metodologías más populares son:

- XP.
- SCRUM.
- ASD.
- CRYSTAL CLEAR.
- AUP.

⁴<http://seccperu.org/files/Metodologias%20Agiles.pdf>

- LSD.

Se realizará una descripción de cada una de estas metodologías, así como de sus características principales.

1.1.6. XP EXTREME PROGRAMMING⁵

Es una metodología de desarrollo de software eficiente, de bajo riesgo y flexible. Se basa en la simplicidad, comunicación y reciclado continuo de código. Además se centra en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promueve el trabajo en equipo, y se preocupa por que sus desarrolladores aprendan y tengan un buen clima de trabajo. Esta metodología fue formulada por Kent Beck en 1999.

Se puede considerar a la programación extrema como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto y aplicarlo de manera dinámica durante el ciclo de vida del software. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. XP se retroalimenta continuamente con la comunicación que existe entre el cliente y el equipo de desarrollo, y todos los participantes. XP promueve la adaptabilidad de los procesos de desarrollo basándose en los principios y prácticas que presenta.

Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre.

⁵<http://www.willydev.net/descargas/prev/TodoAgil.Pdf>

1.1.6.1. Elementos de la metodología XP.

1.1.6.1.1. Historias del usuario.

Es una técnica utilizada para especificar los requisitos del software. Son tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe tener. Es muy comprensible y delimitada de tal forma que el programador pueda implementarla en pocas semanas. Las historias de usuario se descomponen en tareas de programación y se asignan a los programadores para ser implementadas durante una iteración.

1.1.6.2. Roles XP.

- **Programador.-** Es quien produce el código del sistema y escribe las pruebas unitarias.
- **Cliente.-** es quien escribe las historias del usuario y las pruebas funcionales para validar la implementación. Además debe asignar la prioridad a las historias de usuario y decide en que iteración debe ser implementada.
- **Encargado de pruebas (tester).-** Es quien ejecuta las pruebas y luego informa los resultados al equipo, además ayuda al cliente a escribir las pruebas funcionales.
- **Encargado de seguimiento (traker).-** Realiza el seguimiento del progreso de cada iteración y proporciona la realimentación al equipo de trabajo.
- **Entrenador (coach).-** Es el responsable del proceso global. Debe proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.
- **Consultor.-** Es un miembro externo del equipo, quien posee conocimiento en algún tema necesario para el proyecto.

- **Gestor (*bigboss*).**- Es el vínculo entre clientes y programadores, su función principal es la coordinación.

1.1.6.3. Proceso XP.

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El programador construye ese valor.
4. Vuelve al paso 1.

1.1.6.4. Prácticas

XP aplica disciplinadamente de las siguientes prácticas:

- ***El juego de la planificación.*** Existe comunicación constante entre el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.
- ***Entregas pequeñas.*** Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Una entrega no debería tardar más de 3 meses.
- ***Metáfora.*** Es una historia compartida que describe como debe funcionar el sistema. El sistema se define con una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Se usan para dar una visión general y un entendimiento común del sistema y sus funciones y de cómo se debería construir.

- **Diseño simple.** Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.
- **Pruebas.** Éstas son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.
- **Refactorización (refactoring).** Es una actividad constante de reestructuración del código con el objetivo de evitar duplicación del código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo.
- **Programación en parejas.** Toda la producción de código debe realizarse en parejas. Esto conlleva ventajas implícitas (menor tasa de errores, mejor diseño, mayor satisfacción de los programadores...).
- **Propiedad colectiva del código.** Cualquier programador puede cambiar cualquier parte del código en cualquier momento.
- **Integración continua.** Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.
- **40 horas por semana.** Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. El trabajo extra desmotiva al equipo.
- **Cliente in-situ.** El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Éste es uno de los principales factores de éxito del proyecto XP. El cliente conduce constantemente el trabajo hacia lo que aportará mayor valor al negocio y los programadores pueden resolver de manera más inmediata cualquier duda asociada. La comunicación oral es más efectiva que la escrita.

- **Estándares de programación.** XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible.

La mayoría de las prácticas propuestas por XP de alguna forma ya habían sido propuestas en ingeniería del software e incluso demostrado su valor en la práctica. XP las integra de una forma efectiva y las complementa con otras ideas desde la perspectiva del negocio, los valores humanos y el trabajo en equipo.

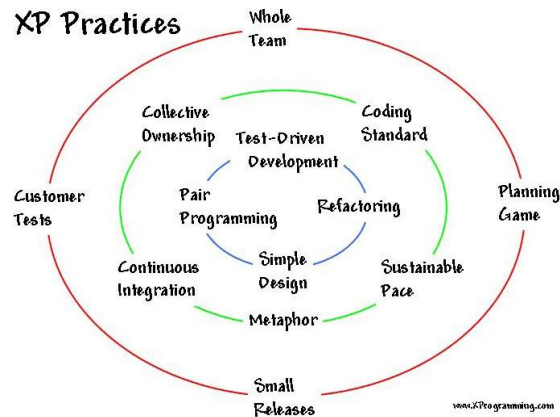


Figura.3 Esquema de trabajo de XP⁶

1.1.6.5. Principio de XP.

Los principios básicos de la programación extrema son:

- **Simplicidad.**- Se simplifica el diseño para agilizar el desarrollo y facilitar el mantenimiento. También se aplica la simplicidad en la documentación, de esta manera el código debe estar autodocumentado. Para ello se deben elegir adecuadamente los nombres de las variables, métodos y clases.

⁶<http://seccperu.org/files/Metodologias%20Agiles.pdf>

- **Comunicación.**- Se realiza de diferentes formas.

Para los programadores el código comunica mejor cuanto más simple sea. Debe comentarse sólo aquello que no va a variar, por ejemplo, el objetivo de una clase o la funcionalidad de un método.

Las pruebas unitarias son otra forma de comunicación ya que describen el diseño de las clases y los métodos al mostrar ejemplos concretos de cómo utilizar su funcionalidad.

Los programadores se comunican constantemente gracias a la programación por parejas.

La comunicación con el cliente es fluida ya que el cliente forma parte del equipo de desarrollo. El cliente decide qué características tienen prioridad y siempre debe estar disponible para solucionar dudas.

- **Retroalimentación (feedback).**- Al trabajar con ciclos cortos de entrega, es muy fácil reconocer las partes que no cumplen con los requisitos y centrarse en lo que es importante. Además como el cliente es parte del equipo de trabajo la opinión que dé del proyecto se conoce en tiempo real. El código también es una fuente de retroalimentación gracias a las herramientas de desarrollo. Por ejemplo, las pruebas unitarias informan sobre el estado de salud del código. Ejecutar las pruebas unitarias frecuentemente permite descubrir fallos debidos a cambios recientes en el código.
- **Coraje o Valentía.**- Para muchos gerentes la programación en parejas no es aceptada ya que parece que la productividad se fuese a reducir a la mitad al estar únicamente una

persona programando. Hay que ser valiente para confiar en que la programación por parejas beneficia la calidad del código sin repercutir negativamente en la productividad.

La simplicidad es uno de los principios más difíciles de adoptar. Se requiere coraje para implementar las características que el cliente quiere ahora sin caer en la tentación de optar por un enfoque más flexible que permite futuras modificaciones.

1.1.6.6. Actividades de XP.

Se describen las actividades que se llevan a cabo en el desarrollo de un proyecto.

1.1.6.6.1. Codificar.

La codificación puede ser dibujar diagramas que generarán código, hacer scripts de sistemas basados en web o codificar un programa que ha de ser compilado.

La codificación también puede usarse para entender la solución más apropiada. La codificación puede ayudar también a comunicar pensamientos sobre problemas de programación. Un programador que trate con un problema de programación complejo y encuentre difícil explicar la solución al resto, podría codificarlo y usar el código para demostrar lo que quería decir. El código, dicen los partidarios de esta posición, es siempre claro y conciso y no se puede interpretar de más de una forma. Otros programadores pueden dar retroalimentación de ese código codificando también sus pensamientos.

1.1.6.6.2. Probar.

Nadie puede estar seguro de algo si no lo ha probado. En el desarrollo de software, XP dice que uno no puede estar seguro de que una función funciona si no la prueba. Esto sugiere la necesidad de definir de lo que uno puede no estar seguro.

- No puedes estar seguro de si lo que has codificado es lo que querías significar. Para probar esta incertidumbre, XP usa pruebas unitarias. Son pruebas automatizadas que prueban el código. El programador intentará escribir todas las pruebas en las que piensa puedan cargarse el código que está escribiendo; si todas las pruebas se ejecutan satisfactoriamente entonces el código está completo.
- No puedes estar seguro de si lo que querías significar era lo que deberías. Para probar esta incertidumbre, XP usa pruebas de aceptación basadas en los requisitos dados por el cliente.

1.1.6.6.3. Escuchar.

Para que los programadores encuentren cual debe ser la funcionalidad del sistema, deben escuchar las necesidades de los clientes. También tienen que intentar entender el problema del negocio y dar a los clientes retroalimentación sobre el problema, para mejorar el propio entendimiento del cliente sobre el problema.

1.1.6.6.4. Diseñar

Desde el punto de vista de la simplicidad, uno podría decir que el desarrollo de sistemas no necesita más que codificar, probar y escuchar. Si estas actividades se desarrollan bien, el resultado debería ser un sistema que funcionase. En la práctica, esto no ocurre. Uno puede seguir sin diseñar, pero un momento dado se va a atascar. El sistema se vuelve muy complejo y las dependencias dentro del sistema dejan de estar claras. Uno puede evitar esto creando una estructura de diseño que organice la lógica del diseño. Buenos diseños evitarán

perdidas de dependencias dentro de un sistema; esto significa que cambiar una parte del sistema no tendrá por qué afectar a otras.

1.1.7. SCRUM⁷

Scrum es un proceso ágil que se puede usar para gestionar y controlar desarrollos complejos de software y productos usando prácticas iterativas e incrementales.

Esta metodología fue desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. El objetivo es elevar al máximo la productividad de un equipo. Es utilizada para proyectos con un rápido cambio de requisitos.

1.1.7.1. Características

Scrum es un esqueleto de proceso que incluye un conjunto de prácticas y roles predefinidos.

Los roles principales en Scrum son:

- **ScrumMaster** que mantiene los procesos y trabaja junto con el jefe de proyecto.
- **ProductOwner** que representa a las personas implicadas en el negocio.
- **Team** que incluye a los desarrolladores.

La metodología propone que se haga una lista con todas los requisitos de trabajo de alto nivel que debe tener el sistema y ordenarlas de acuerdo a como serán desarrollados. Esta lista es conocida como "ProductBacklog". Quien quiera puede añadirle funcionalidades a la lista ya que no todas estarán puestas inicialmente. La persona encargada de ordenar el "ProductBacklog" y de lo que se va a hacer en la aplicación es conocido como

⁷agil.pdf.- Instituto Nacional de Tecnologías de la Comunicación

"ProductOwner", además informa al equipo cuales items del "ProductBacklog" deben cambiarse.

Las funcionalidades elegidas se descomponen en tareas concretas y se les asigna un tiempo para hacerlas y una persona. La lista de tareas se debe hacer en un periodo de 2 a 4 semanas es decir en una iteración conocida como "Sprint Backlog". Los ítems que entran en una iteración se determinan durante la reunión de planificación. Nadie puede cambiar el backlog, lo que significa que los requisitos están congelados para esa iteración. Cuando se completa una iteración, el equipo demuestra el uso del software.

Al fin de cada iteración hay una demostración a cargo del Scrum Master. Las presentaciones no se deben realizar en PowerPoint. Es permitido usar artefactos de los métodos a los que Scrum acompañe.

Otra característica importante son las reuniones a lo largo del proyecto. La reunión diaria de 15 minutos del equipo de desarrollo es para coordinación e integración, en la que todos cuentan qué han hecho ayer, qué van hacer hoy y qué problemas tienen para hacer lo que están haciendo. En los encuentros diarios, todos tienen que ser puntuales; si alguien llega tarde, se le cobra una multa que se destinará a obras de caridad.

Hay distintos tipos de reuniones:

- **Scrum diario:** cada día durante la iteración, tiene lugar una reunión de estado del proyecto. A esta reunión se le domina Scrum
- **Reunión de planificación de iteración (sprint):** se lleva a cabo al principio del ciclo de la iteración.
- **Reunión de revisión de iteración:** al final del ciclo de la iteración.

- **Iteración retrospectiva:** al final del ciclo de la iteración.

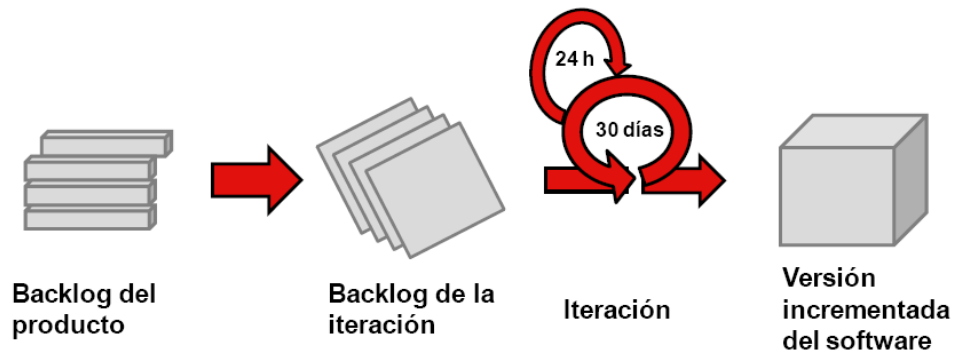


Figura.4 Flujo del proceso de SCRUM

1.1.7.2. PRÁCTICAS DE SCRUM

A continuación se enumeran algunas de las prácticas de Scrum:

- Los clientes se convierten en parte del equipo de desarrollo.
- Frecuentes entregables intermedios que funcionen, como otras formas de procesos de software ágiles. El cliente trabaja con el software antes, además permite que el proyecto cambie los requisitos de acuerdo con las necesidades.
- El equipo de desarrollo debe hacer planes de riesgos y mitigación frecuentes, la mitigación de riesgos, la monitorización y la gestión de riesgos se lleva a cabo en todas las etapas y con compromiso.
- Transparencia en la planificación y desarrollo de módulos, esto permite a cada persona saber quién es responsable de qué y cuándo.
- Reuniones frecuentes de las personas involucradas en el negocio, permite monitorizar el progreso.
- Debe existir un mecanismo de advertencias avanzado.

- Los problemas no se deben ocultar. Nadie es penalizado por reconocer o describir un problema imprevisto.

1.1.7.3. CICLO DE VIDA DE SCRUM

El ciclo de vida de Scrum es el siguiente:

1. **Pre-Juego: Planeamiento.** El propósito es establecer la visión, definir expectativas y asegurar la financiación. Las actividades están formadas por:

- La escritura de la visión.
- El presupuesto.
- El registro de acumulación o retraso (backlog) del producto inicial. Este debe ser de alto nivel de abstracción.
- Los ítems estimados
- La arquitectura de alto nivel.
- El diseño exploratorio y los prototipos.

2. **Pre-Juego: Montaje (Staging).** El propósito es identificar más requerimientos y priorizar las tareas a realizar en la primera iteración. Las actividades son:

- Planificación.
- Diseño exploratorio y
- Prototipos.

3. **Juego o Desarrollo.** El propósito es implementar un sistema listo para entregar, en varias iteraciones de treinta días llamadas “corridas” (sprints). Las actividades son:

- Un encuentro de planeamiento de corridas en cada iteración.

- La definición del registro de acumulación de corridas y los estimados.
- Encuentros diarios de Scrum.

4. **Pos-Juego: Liberación.** El propósito es el despliegue operacional. Las actividades son:

- Documentación.
- Entrenamiento.
- Mercadeo.
- Venta.

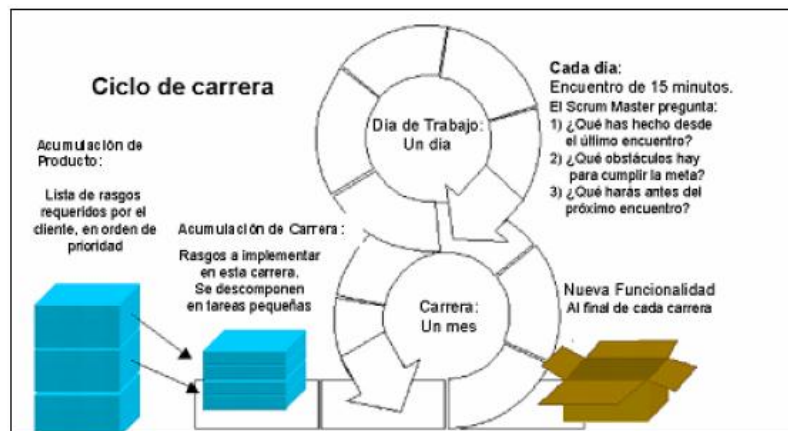


Figura.5 Ciclo de vida de Scrum⁸

Una de las mayores ventajas de Scrum es que es muy fácil de entender y requiere poco esfuerzo para comenzar a usarse.

En Scrum, el equipo se focaliza en una única cosa: construir software de calidad.

Por otro lado, la gestión de un proyecto Scrum se focaliza en definir cuáles son las características que debe tener el producto a construir y en remover cualquier obstáculo que pudiera entorpecer la tarea del equipo de desarrollo.

⁸ http://tratandodeentenderlo.blogspot.com/2010_06_01_archive.html

En esta metodología el cliente se entusiasma y se compromete con el proyecto dado que ve crecer el producto iteración a iteración y encuentra las herramientas para alinear el desarrollo con los objetivos de negocio de su empresa. Por otro lado, los desarrolladores encuentran un ámbito propicio para desarrollar sus capacidades profesionales y esto resulta en un incremento en la motivación de los integrantes del equipo.

Scrum nos indica cómo conseguir que todos trabajen con el mismo objetivo, a corto plazo y deja bastante visible como avanza el proyecto día a día. Además define métodos de gestión y control para complementar la aplicación de otros métodos ágiles como XP o alguna otra metodología de desarrollo.

1.1.8. ASD – ADAPTIVE SOFTWARE DEVELOPMENT⁹

El método ágil ASD (Adaptive Software Development) traducido en español significa Desarrollo Adaptable de Software es un modelo de implementación de patrones ágiles para desarrollo de software. Su funcionamiento es cíclico y reconoce que en cada iteración se producirán cambios e incluso errores.

Fue desarrollado en el año 2000 por James Highsmith, consultor de CutterConsortium.

Este método ágil pretende abrir una tercera vía entre el “desarrollo monumental de software” y el “desarrollo accidental”, o entre la burocracia y la adhocracia.

Se basa en el concepto de emergencia, el cual describe la forma en que la interacción de las partes genera una propiedad que no puede ser explicada en función de los componentes individuales.

⁹ schenone-tesisdegradoingenieriainformatica.pdf

Para Highsmith, los proyectos de software son sistemas adaptativos complejos y la optimización no hace más que sofocar la emergencia necesaria para afrontar el cambio. En los sistemas complejos no es aplicable el análisis, porque no puede deducirse el comportamiento del todo a partir de la conducta de las partes, ni sumarse las propiedades individuales para determinar las características del conjunto.

1.1.8.1. CARACTERÍSTICAS DE ASD

Sus principales características son:

- Iterativo.
- Orientado a los componentes de software más que a las tareas .
- Tolerante a los cambios.
- Guiado por los riesgos.

La revisión de los componentes sirve para aprender de los errores y volver a iniciar el ciclo de desarrollo.

ASD presupone que las necesidades del cliente son siempre cambiantes. La iniciación de un proyecto involucra definir su misión, determinar las características y las fechas, y luego descomponer el proyecto en una serie de pasos individuales, cada uno de los cuales puede comprender entre cuatro y ocho semanas. Los pasos iniciales deben verificar el alcance del proyecto; los tardíos tienen que ver con el diseño de una arquitectura, la construcción del código, la ejecución de las pruebas finales y el despliegue.

1.1.8.2. ASPECTOS CLAVES DE ASD

Los aspectos claves de la metodología ASD son:

1. Un conjunto no estándar de “artefactos de misión” (documentos), incluyendo una visión del proyecto, una hoja de datos, un perfil de misión del producto y un esquema de su especificación
2. Un ciclo de vida, inherentemente iterativo.
3. Cajas de tiempo, con ciclos cortos de entrega orientados por riesgo.

Un ciclo de vida es una iteración; este ciclo se basa en componentes y no en tareas, es limitado en el tiempo, orientado por riesgos y tolerante al cambio. Al basarse en componentes implica concentrarse en el desarrollo de software, construyendo el sistema pieza por pieza. Entonces, el cambio es bienvenido y necesario, pues es la oportunidad de aprender y ganar una ventaja competitiva.

1.1.8.3. CICLO DE VIDA DE ASD

ASD utiliza un "cambio orientado hacia el ciclo de vida", que tiene tres componentes que son:

1.1.8.3.1. Especular

Se establecen los principales objetivos y metas del proyecto en su conjunto, se debe comprender las limitaciones (zonas de riesgo) con las que operará el proyecto. Se realizan estimaciones de tiempo sabiendo que pueden sufrir desviaciones. Además se decide el número de iteraciones para terminar el proyecto, prestando atención a las características que pueden ser utilizadas por el cliente al final de la iteración. Por tanto son necesarios, marcar objetivos prioritarios dentro de las mismas iteraciones.

Estos pasos se puede volver a examinar varias veces antes de que el equipo y los clientes estén satisfechos con el resultado.

1.1.8.3.2. Colaborar

En esta fase se centra la mayor parte del desarrollo manteniendo una componente cíclica. Es muy importante que exista la coordinación, ya que un equipo debe transmitir a los otros equipos lo que aprendieron, para que estos no vuelvan a aprender.

1.1.8.3.3. Aprender

Esta etapa tiene una serie de ciclos de colaboración, se debe capturar lo que se ha aprendido, tanto positivo como negativo. Es un elemento crítico para la eficacia de los equipos.

JimHighsmith identifica cuatro tipos de aprendizaje en esta etapa:

- 1. *Calidad del producto desde un punto de vista del cliente.*** Los clientes tienen un valor importante ya que es la única manera de saber si se ha tenido éxito.
- 2. *Calidad del producto desde un punto de vista de los desarrolladores.*** Se realiza una evaluación de la calidad de los productos desde un punto de vista técnico.
- 3. *La gestión del rendimiento.*** Se evalúa lo que se ha aprendido mediante el empleo de los procesos utilizados por el equipo.
- 4. *Situación del proyecto.*** Es el punto de partida para la construcción de la siguiente serie de características, es decir es un paso previo a la planificación de la siguiente iteración del proyecto.

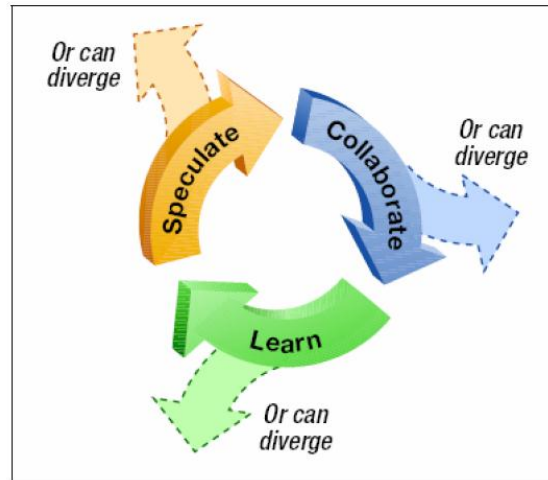


Figura.6 El ciclo de vida adaptativo. Tomada de [Highsmith, 2000].

Al usar la metodología ASD se puede alcanzar excelentes resultados, pero debido a las características que maneja es más factible usarla para proyectos pequeños y medianos, para adquirir práctica y experiencia.

Además ASD reconoce que el desarrollo de software es un proceso vivo, que no puede regirse por procesos centrados en los productos.

Recuerde que el cambio, cualquiera que sea, se ve como algo negativo. Se le debe introducir en la organización con mucho tacto. El éxito depende de que *todo* el equipo crea en él.

1.1.9. AUP

AUP Agile UnifiedProcess o elProceso Unificado Agil de Scott Ambler, es una versión simplificada del Proceso Unificado de Rational (RUP). Describe de manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP.

AUP aplica técnicas ágiles incluyendo Desarrollo Dirigido por Pruebas (test drivendevelopment - TDD), Modelado Ágil, Gestión de Cambios Ágil, y Refactorización de Base de Datos para mejorar la productividad.¹⁰

Scott Ambler, dijo:

"Si buscas algo entre Extreme Programming y RUP tradicional, un proceso que es ágil pero incluye explícitamente actividades y artefactos con los que estás acostumbrado, entonces AUP, Agile UnifiedProcess, es para tí".

1.1.9.1. FASES DE AUP.¹¹

AUP es capturada en 4 fases:

1. **Incepción.**- El objetivo es identificar el alcance inicial del proyecto, una arquitectura potencial para el sistema, obtener el financiamiento y la aceptación de los stakeholders.
2. **Elaboración.** El objetivo es proveer la arquitectura del sistema.
3. **Construcción.** El objetivo es desarrollar software trabajando de forma regular, se basa en incrementos que satisface las necesidades de mayor prioridad de los stakeholders del proyecto.
4. **Transición.** El objetivo es validar y deployar el sistema en el entorno de producción.

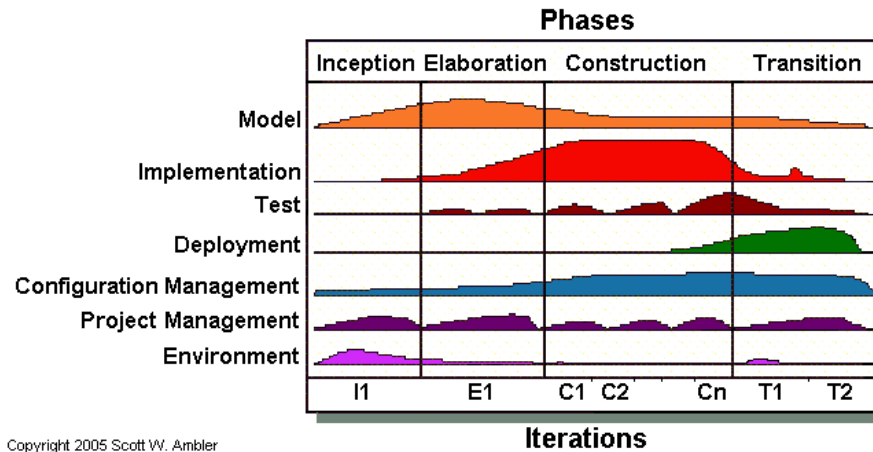
¹⁰http://en.wikipedia.org/wiki/Agile_software_development

¹¹<http://www.ambysoft.com/unifiedprocess/agileUP.html>

1.1.9.2. DISCIPLINAS DE AUP

A diferencia de RUP, AUP tiene solamente siete disciplinas:

1. **Modelo.** Entender el negocio de la organización, el problema que es tratado en el proyecto, e identificar una solución viable para tratar el problema.
2. **Puesta en práctica.** Los modelos son transformados en código ejecutable, se debe realizar un nivel básico de pruebas, es decir prueba de la unidad.
3. **Prueba.** La evaluación debe ser objetiva para asegurar calidad. Esto incluye encontrar defectos, validar que el sistema trabaja según lo diseñado, y verificar que los requisitos se cumplan.
4. **Despliegue.** Se debe realizar un plan de entrega del sistema y luego ejecutarlo para poner el sistema a disposición de los usuarios finales.
5. **Gerencia de la configuración.** Maneja el acceso a los artefactos del proyecto. Incluye no sólo seguir versiones del artefacto en un cierto plazo, sino también controlarlas y manejarlas.
6. **Gerencia de proyecto.** Las actividades que ocurren dentro del proyecto son dirigidas, como son: riesgos de manejo, dirección y coordinación con personas (asignación de tareas, seguimiento de progreso, etc.), y el alcance del proyecto para estar seguro que se entrega a tiempo y dentro del presupuesto.
7. **Ambiente:** Apoyar el resto de los esfuerzos para garantizar que el proceso es adecuado, además que la orientación (normas y directrices), y herramientas (hardware, software, etc) estén disponibles para el equipo cuando lo necesiten.



Copyright 2005 Scott W. Ambler

Figura.7 Disciplinas

1.1.9.3. FILOSOFÍAS DE AUP

1. *El personal sabe lo que él está haciendo.* La gente no lee la documentación detallada del proceso, ella desea cierta dirección de alto nivel y/o entrenamiento de vez en cuando. AUP proporciona acoplamiento a muchos de los detalles, si desea, pero no los fuerza a utilizarlos.

2. *Simplicidad.* Se debe utilizar una cantidad precisa de páginas, no millares de ellas.

3. *Agilidad.* El ascendente ágil se conforma con los valores y los principios del desarrollo ágil del software y Alianza ágil.

4. *Foco en actividades de alto valor.* Son las actividades que realmente son importantes, no cada cosa que puede sucederle al proyecto.

5. *Independencia de la herramienta.* Se puede utilizar cualquier toolset que desee. La recomendación es utilizar herramientas adecuadas para el trabajo, que son a menudo herramientas simples.

6. *Usted deseará adaptar el AUP para resolver sus propias necesidades.* AUP es fácil de manejar con cualquier herramienta de edición de HTML. No necesita comprar una herramienta especial, o tomar un curso, para adaptar AUP.

1.1.9.4. RELEASE

AUP tiene dos tipos de iteraciones.

- **Release de desarrollo** da lugar a un despliegue al área de la garantía y/o de la versión parcial de programa de calidad.
- **Release de producción** da lugar a un despliegue al área de la producción.



Figura.8 Tipos de release

El primer release generalmente toma más tiempo que los demás, ya que el equipo de trabajo tiene la necesidad de obtener la mayor parte del sistema, al existir la colaboración del equipo. La primera producción de liberación puede tomar doce meses, la segunda nueve meses y las otras cada seis meses.¹²

1.1.10. LSD LEAN PRODUCT DEVELOPMENT¹³

Esta metodología de desarrollo de softwarees una translación de los principios y prácticas de la manufacturación, estos principios son aplicables a cualquier ambiente para mejorar el desarrollo de software.

Fue creada por Robert Charette y luego popularizada por Mary y Tom Poppendieck.

Su origen es parte del Toyota ProductionSystem, es una filosofía de trabajo similar a la *Calidad Total* y el *Just-In-Time*, pero trata de disminuir los costos y aumentar los ingresos al reducir el desperdicio de tiempo y horas/hombre durante el proceso de manufactura.

¹²http://en.wikipedia.org/wiki/Agile_software_development

¹³<http://everac99.spaces.live.com/blog/cns!B2296C467C188917!659.entry.htm>

Entonces el objetivo principal es:

Desarrollar software en un tercio de tiempo, con un tercio del presupuesto y con un tercio de la tasa de defectos.

1.1.10.1. PRINCIPIOS¹⁴

Los principios Lean, aplicados al desarrollo de software, son:

Eliminar los desperdicios

Se apoya en técnicas que ayuden a prevenir "trabajo parcialmente terminado", o lo que es lo mismo: documentación no implementada y código no sincronizado, sin probar, documentar o deployar.

Se invierte tiempo solo en las actividades y recursos que agregan valor real al proyecto. El determinar qué es y qué va a agregar valor se hace en los niveles altos de la jerarquía. Con el fin de poder eliminar los residuos, se debería ser capaz de reconocerlo y encontrarlo.

Los principales tipos de desperdicios son:

- Funcionalidad que se desarrolla y no se utiliza en producción.
- La documentación.
- Asignar una misma persona a múltiples proyectos.
- Las Esperas.
- Puesta en marcha.
- Errores en el Software.

¹⁴<http://blog.tercerplaneta.com/2007/05/lean-product-development.html>

Para distinguir y reconocer los residuos se utiliza una técnica llamada *valuestreammapping* (o *mapa de flujo de valor*). El siguiente paso consiste en señalar las fuentes de los residuos y eliminarlos.

Ampliar el aprendizaje

Se debe incrementar el feedback mediante reuniones cortas con los clientes para saber con exactitud lo que desea y resolver los problemas complicados que se presenten y buscar una solución para mejorar el desarrollo. La acumulación de defectos debe evitarse ejecutando las pruebas tan pronto como el código está escrito. En lugar de añadir más documentación o planificación detallada, las distintas ideas podrían ser probadas escribiendo código e integrándolo.

Por lo tanto, los clientes comprenden mejor sus necesidades, basándose en el resultado de los esfuerzos del desarrollo, y los desarrolladores aprendan a satisfacer mejor estas necesidades.

Decidir lo más tarde posible

El desarrollo de software tiene cierto grado de incertidumbre, lo que hace que las decisiones se retrasen, una vez que se sabe lo que desea que haga el proyecto, se puede *tomar una decisión la cual será tomada en el momento de mayor certeza*, esta estará basada en un conocimiento concreto, no en especulación, una vez que los clientes hayan reconocido mejor sus necesidades.

La idea es esperar hasta el momento en que se encuentre disponible la mejor y mayor información. De esta forma, se evitarán cambios en la etapa final del desarrollo, y se bajarán los costos del mismo.

Entregar lo más temprano posible

Al entregar el producto final sin defectos considerables cuanto antes, más pronto se pueden recibir comentarios y agregarlos en la siguiente iteración. Cuanto más cortas sean las iteraciones, mejor es el aprendizaje y la comunicación dentro del equipo.

Hacer entregas de funcionalidad rápidamente tiene muchas ventajas:

- El usuario tiene la funcionalidad que necesita cuando la necesita.
- Se tiene un feedback más confiable por parte del usuario.
- Se puede retrasar la toma de decisiones.
- Es más claro, para el equipo de trabajo, de qué forma contribuir más efectivamente.

Dar poder al equipo

Al considerar a las personas lo más importante dentro del desarrollo de software, ya que ellos son quienes conocen los detalles, los métodos de formación de equipos son focalizados para poder resolver sus propios problemas. Al involucrar a cada persona en la toma de decisiones, se logra que cada integrante del equipo se sienta partícipe del proyecto y la organización, por lo tanto usarán su potencial al máximo. Las personas necesitan motivación y un propósito superior para el cual trabajar; un objetivo alcanzable dentro de la realidad, con la garantía de que el equipo puede elegir sus propios compromisos. El líder del equipo se encarga de marcar el objetivo, establecer ciertas pautas de trabajo, ser el guía, proporcionar el apoyo y ayuda en situaciones difíciles. Además los desarrolladores deben tener acceso a los clientes.

Construir Integridad

Tenemos 2 clases de integridad: Perceptiva, Conceptual.

La integridad perceptiva se refiere a la percepción que tiene el cliente del sistema, si se acerca más a lo que desea o a sus exigencias y requerimientos, si es así más alta será la integridad perceptiva.

La integridad conceptual significa que los componentes separados del sistema funcionen bien juntos, como en un todo, logrando equilibrio entre la flexibilidad, mantenibilidad, eficiencia y capacidad de respuesta.

Otro nivel de integridad está relacionado con la adaptabilidad del mismo en el futuro. Un software con integridad tiene una arquitectura coherente, se ajusta a los propósitos.

La integridad se puede lograr con un sabio liderazgo, comunicación efectiva y una disciplina saludable.

Ver el Todo

Los sistemas de software hoy en día no son simplemente la suma de sus partes, sino también el producto de sus interacciones.

La integridad, en sistemas complejos, requiere una profunda experiencia en diversas áreas. Uno de los problemas más difíciles a tratar en el desarrollo de un producto es que los expertos en alguna área específica tienden a maximizar la performance de la parte del producto que representan, en lugar de hacer énfasis en la performance del sistema en su conjunto. Esto se denomina Sub-Optimización. El desafío es implementar prácticas que la eliminen.

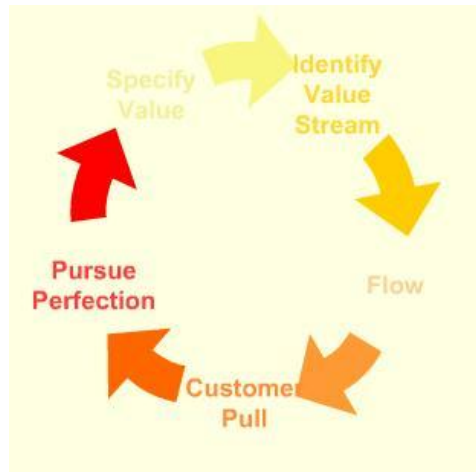


Figura.9 Principios

1.2. METODOLOGÍAS TRADICIONALES DE DESARROLLO DE SOFTWARE

Inicialmente el desarrollo de software no tenía un proceso formal, es así que se adaptaron metodologías existentes de otras áreas. Esta adaptación dividió el desarrollo de software en etapas secuenciales que en cierta manera solucionó la necesidad del área de software.

Así surgen las metodologías de desarrollo llamadas Tradicionales o Pesadas, las cuales tienen mayor énfasis en la planificación y control del proyecto.

Las metodologías tradicionales se centran en una disciplina de trabajo sobre el proceso de desarrollo del software, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada, para así conseguir software más eficiente. En la planificación se detalla todo lo requerido, una vez hecho esto, comienza el ciclo de desarrollo del producto de software, ya que si luego se desea implementar un cambio su coste es alto.

Las metodologías tradicionales centran su atención en llevar una documentación exhaustiva de todo el proyecto, todo esto definido en la fase inicial del desarrollo del software.¹⁵

Su premisa fundamental argumenta que hay suficiente planeación y administración, por lo que el resultado puede predecirse y así mismo pueden evitarse los riesgos.

Uno de los inconvenientes de las metodologías tradicionales es que no se adaptan adecuadamente a los cambios, por lo que no son métodos adecuados cuando se trabaja en un entorno, donde los requisitos no pueden predecirse o pueden variar.

Entonces se dice que las metodologías tradicionales tienen las siguientes características:

- Se basa en documentos.
- Se elaboran definiciones Flujo Trabajo.
- Existen muchos roles diferentes.
- Muchos puntos de control.
- Tiene alto sobrecoste de gestión.
- Mucha burocracia.

Las metodologías tradicionales se clasifican en:

- **Estructuradas**.- se basan en el enfoque top-down
 - ✓ *Orientadas a procesos*.- su base es el modelo entrada/proceso/salida y por diagrama de flujo de datos (DFD), diccionario de datos y especificaciones de proceso.

¹⁵Articulo TecnicoMetodologias Desarrollo.doc

- ✓ *Orientadas a datos.*- se basan en la información, define primero la estructura de datos de las q se deriva los componentes procedimentales,
- ✓ *Mixtas.*- combinan la orientación a procesos y a datos.
- *No estructuradas.*
 - ✓ Orientada a objetos.- encapsula procesos y datos en el concepto de objeto.
 - ✓ Sistemas de tiempo real.- procesa información orientada más al control q a los datos. se caracterizan por concurrencia, priorización de procesos, comunicación entre tareas y acceso simultáneo de datos comunes.

1.2.1. EJEMPLOS DE METODOLOGÍAS TRADICIONALES.

A continuación algunos ejemplos de metodologías tradicionales:

1.2.1.1. BASADAS EN CICLOS DE VIDA.

Tenemos las siguientes metodologías basadas en el ciclo de vida.

1.2.1.1.1. EN CASCADA

Su característica principal es que no comienza una fase hasta que no ha terminado la anterior.

1.2.1.1.2. PROTOTIPADO RÁPIDO

Consiste en iterar en la fase de análisis tantas veces como sea necesario, mostrando prototipos al usuario para que pueda indicar de forma más eficiente los requisitos del sistema.

1.2.1.1.3. EVOLUTIVO

Se asume que los requisitos pueden cambiar en cualquier momento del ciclo de vida y no solo en la etapa de análisis. Se repite todo el ciclo para desarrollar nuevas versiones de todo el sistema.

1.2.1.1.4. INCREMENTAL

Se desarrolla un subsistema para satisfacer un subconjunto de los requisitos especificados y en posteriores versiones se añaden nuevas funcionalidades que satisfagan más requisitos.

1.2.1.1.5. EN ESPIRAL

Toma las ventajas del modelo de desarrollo en cascada y el de prototipos, incorporando el concepto de análisis de riesgo en cada iteración.

1.2.1.2. METODOLOGÍAS ORIENTADAS A PROCESOS

Metodologías de desarrollo en la que se definen un conjunto de buenas prácticas para la mejora gradual de los procesos de ingeniería.

Algunos ejemplos de metodologías orientadas a procesos y su descripción:

1.2.1.2.1. RationalUnifiedProcess (RUP)

Define un ciclo de vida iterativo priorizando el uso de lenguajes de modelado, casos de uso y centrado en la arquitectura.

1.2.1.2.2. Microsoft Solution Framework (MSF)

Metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso. Se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.

En la siguiente tabla se puede observar otros ejemplos de metodologías orientadas a procesos:

- *CapabilityMaturityModel (SW-CMM)*
- *CapabilityMaturityModelIntegrationforDevelopment (CMMI-DEV)*
- *Big Design Up Front (BDUF)*
- *Cleanroom Software Engineering*
- *RationalUnifiedProcess (RUP)*
- *EssentialUnifiedProcessfor Software Development (EssUP)*
- *FuseboxLifecycleProcess (FLiP)*
- *Software ProcessImprovement and Capabilitydetermination (SPICE)*
- *Métrica*
- *Jackson SystemDevelopment (JSD)*
- *JointApplicationDevelopment (JAD)*
- *Open UnifiedProcess (OpenUP)*

Figura.10 Metodologías orientadas a procesos

1.3. COMPARACIONES DE LAS METODOLOGÍAS TRADICIONALES Y METODOLOGÍAS ÁGILES.

A continuación se muestra una tabla comparativa entre las metodologías tradicionales y las metodologías ágiles de desarrollo de software. Esta tabla recoge esquemáticamente las principales diferencias de las metodologías ágiles con respecto a las tradicionales (“no

ágiles”). Estas diferencias afectan no sólo al proceso en sí, sino también al contexto del equipo así como a su organización.

Metodologías ágiles	Metodologías tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo(además in-situ)	El cliente interactúa con el equipo de desarrollo mediante reuniones
Menos énfasis en la arquitectura del software, se va definiendo y mejorando a lo largo del proyecto	La arquitectura del software es esencial y se expresa mediante modelos, se define tempranamente en el proyecto
Pocos Artefactos. El modelado es prescindible, modelos desechables.	Más Artefactos. El modelado es esencial, mantenimiento de modelos
Pocos Roles, más genéricos y flexibles	Más Roles, más específicos
Cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Orientada a proyectos pequeños. Corta duración (o entregas frecuentes), equipos pequeños (< 10 integrantes) y trabajando en el mismo sitio	Aplicables a proyectos de cualquier tamaño, pero suelen ser especialmente efectivas/usadas en proyectos grandes y con equipos posiblemente dispersos
Énfasis en los aspectos humanos: el individuo y el trabajo en equipo	Énfasis en la definición del proceso: roles, actividades y artefactos
Se esperan cambios durante el proyecto	Se espera que no ocurran cambios de gran impacto durante el proyecto

Tabla. 1 Diferencias entre Metodologías ágiles y metodologías tradicionales.¹⁶

Entonces se puede decir que las metodologías tradicionales presentan los siguientes problemas a la hora de abordar proyectos:

¹⁶<http://seccperu.org/files/Metodologias%20Agiles.pdf>

- Existen costosas fases previas de especificación de requisitos, análisis y diseño.
- La corrección durante el desarrollo de errores introducidos en estas fases será costosa, es decir, se pierde flexibilidad ante los cambios.
- El proceso de desarrollo está ajustado por documentos firmados.
- El desarrollo es más lento.
- Es difícil para los desarrolladores entender un sistema complejo en su globalidad.

En la figura se muestra un enfoque que tienen las metodologías rigurosas y ágiles.

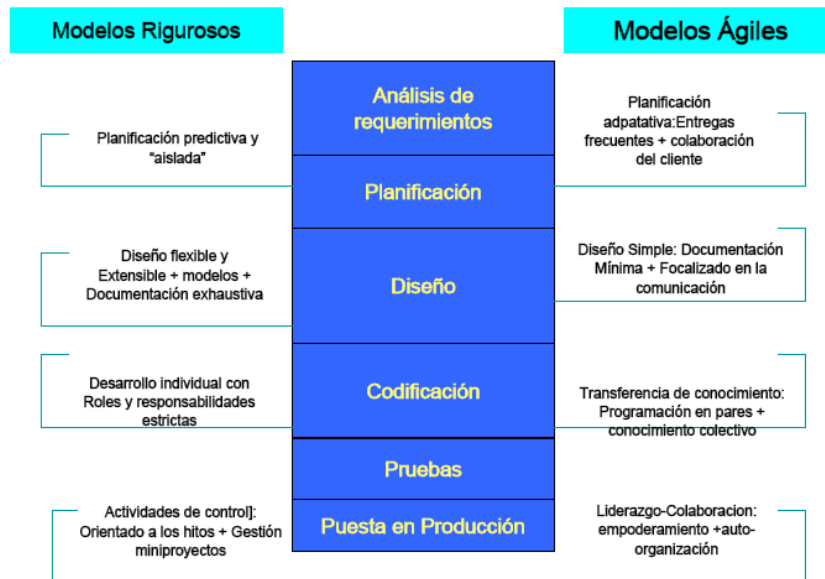


Figura.11 Enfoque de las metodologías

A continuación se muestran algunos de los elementos que pueden caracterizar a las metodologías ágiles:

- Baja criticidad.
- Desarrolladores seniors.
- Los requisitos cambian con mucha frecuencia.

- Pequeño número de desarrolladores.
- Cultura que prospera al caos.

Y para el caso de algunas metodologías tradicionales que son orientadas a planes, se pueden observar los siguientes elementos:

- Alta criticidad
- Desarrolladores juniors
- Los requisitos no cambian con mucha frecuencia
- Gran número de desarrolladores
- Cultura que demanda orden

1.4. VENTAJAS DEL USO DE METODOLOGÍAS ÁGILES

- Las metodologías ágiles reducen el tiempo de desarrollo significativamente y por ende su coste; ya que en la etapa de planificación no intenta definir todos los requerimientos que debe tener el proyecto, si no que se definen los requerimientos principales para realizar el lanzamiento de un prototipo.
- En el desarrollo del software se encuentran involucrados tanto desarrolladores como directivos y clientes, teniendo reuniones cara a cara para definir el funcionamiento del sistema, haciendo que se fomente el trabajo en equipo y se cree mayor responsabilidad en todos los implicados.
- Al no ser una planificación estricta el desarrollo del software, responde a cambios que surjan durante el ciclo de vida del proyecto; tanto de negocio como de tecnología, permitiendo que se adapte sin que afecte en su funcionamiento.

- Solo se hace la documentación necesaria, la cual se utiliza ya sea por los desarrolladores o por los directores, para que luego no se la traspapele y no se la utilice.
- Permite una continua retroalimentación sobre los errores, al finalizar cada iteración logrando que mejore los procesos y el equipo de desarrollo.
- Entrega continua y en plazos breves de software funcional.
- Evita malentendidos de requerimientos entre el cliente y el equipo.
- Apropiado para entornos volátiles.
- Estar preparados para el cambio, significa reducir su coste.
- Planificación más transparente para nuestros clientes, conocen las fechas de entrega defuncionalidades vitales para su negocio.
- Permitirá definir en cada iteración cuales son los objetivos de la siguiente iteración.
- Permite tener realimentación de los usuarios muy útil.
- La presión esta a lo largo de todo el proyecto y no en una entrega final.
- Capacidad de respuesta a cambios de requisitos a lo largo del desarrollo.
- Entrega continua y en plazos breves de software funcional.
- Trabajo conjunto entre el cliente y el equipo de desarrollo.
- Importancia de la simplicidad, eliminado el trabajo innecesario.
- Atención continua a la excelencia técnica y al buen diseño.
- Mejora continua de los procesos y el equipo de desarrollo

1.5. CONCLUSIONES

- El retrasar las decisiones en un proyecto de software permite potenciar el valor del productotanto para el cliente como al equipo o empresa que desarrolla.
- Para que un grupo de desarrollo adopte una metodología ágil debe poseer experienciabajando con metodologías tradicionales, ya que la experiencia es la que predomina en losmomentos cruciales del proyecto, además debe tener la capacidad de ser equiposautogestionados, altamente motivados y con gran innovación.
- Las metodologías ágiles permiten disminuir costos y brindar flexibilidad a los proyectos desoftware donde la incertidumbre está presente.

2. Capítulo II

2.1. METODOLOGÍA DE PROGRAMACIÓN EXTREMA XP.

Un proceso ligero puede ser definido como... *El mínimo conjunto de actividades y elementos que deben ser incluidos en el proceso de desarrollo de Software para asegurar un buen resultado para todas las partes interesadas.*

2.1.1. POR QUE UTILIZAR XP?

Muchas veces nos preguntaremos cuando usar XP. Los proyectos con requerimientos dinámicos son perfectos para XP. Estos proyectos experimentarán grandes éxitos y productividad del desarrollador.

XP es un nuevo concepto refrescante. XP tiene éxito porque da énfasis al involucramiento del cliente y promueve el trabajo del equipo.

El aspecto más sorprendente de XP son sus reglas simples y prácticas. Al inicio parecen torpes y quizás incluso ingenuo, pero pronto se vuelve un cambio bienvenido. A los clientes les gusta estar en el proceso de desarrollo de software y contribuir con los diseñadores activamente sin tener en cuenta el nivel de experiencia.

2.1.2. DEFINICIÓN DE PROGRAMACIÓN EXTREMA O EXTREME PROGRAMMING XP

A continuación tenemos algunas definiciones para la metodología XP dadas por varios autores.

Según Kent Beck quien es considerado como el padre de la metodología XP, menciona que es un proceso ligero, de bajo riesgo, flexible, predecible, científico y divertido de desarrollar software.

Según Wikipedia.

La programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Se puede considerar la programación extrema como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del software.

De acuerdo a Programacionextrema.org

La Programación Extrema es uno de los llamados procesos o metodologías ágiles de desarrollo de software. Consiste en un conjunto de prácticas que a lo largo de los años han demostrado ser las *mejores* del desarrollo de software, llevadas al extremo, fundamentadas en un conjunto de valores”.

Deigote’s Blog menciona:

La programación extrema es una metodología de ingeniería de software para el desarrollo del mismo, que hace énfasis en los siguientes aspectos: satisfacción del cliente y trabajo en equipo.

2.1.3. CONTEXTO XP

La metodología XP tiene el siguiente contexto.

- Cliente bien definido y en colaboración constante.
- Los requisitos pueden y van a cambiar (volátiles).
- Reduce los tiempos de desarrollo manteniendo la calidad.
- Desarrollo incremental y continuo para responder a los cambios.
- Grupo pequeño y muy integrado.

2.1.4. AXIOMAS DE XP

Axioma de XP	Consecuencias
Equipos de pequeñas dimensiones	Equipos de 4 a 8 representan proyectos que son subconjuntos de un proyecto de desarrollo.
Requisitos vagos	En el mundo de los negocios, los requisitos vagos son un indicativo de problemas sistemáticos mayores. Si los resultados del negocio son vagos, el proceso de medida del valor puede no ser estable.
Requisitos cambiantes	Los proyectos cuyos requisitos cambian a menudo tienen también otros problemas. Las partes implicadas no pueden decidir cuál debe ser el resultado, y por tanto la medida del éxito del negocio posiblemente sea también inestable.

Tabla. 2 Axiomas de XP

2.1.5. DERECHOS DE CLIENTES Y DESARROLADORES EN XP¹⁷

2.1.5.1. Derechos del Cliente

Como Cliente tiene derecho a:

- Un plan global, para saber qué puede obtenerse, cuando y a qué coste.
- Obtener el mayor valor posible de cada semana de programación.
- Ver el progreso en un sistema que corre, demostrando que funciona pasando las pruebas repetidas que se especifique (de regresión).
- Cambiar de opinión, para sustituir funcionalidad y cambiar prioridades sin pagar costes desorbitados.
- Estar informado de los cambios de planificación, a tiempo para escoger cómo reducir el alcance y mantener la fecha inicial o incluso cancelar en cualquier momento y quedarse con un sistema útil que funcione y sea reflejo de la inversión realizada hasta la fecha.

2.1.5.2. Derechos del Desarrollador

Como Desarrollador tiene derecho a:

- Conocer lo que se necesita, con definiciones claras de prioridad.
- Producir siempre trabajo de calidad.
- Solicitar y recibir ayuda de los compañeros, superiores y clientes.
- Hacer y actualizar sus propias estimaciones.
- Aceptar sus responsabilidades en lugar de que le sean asignadas.

¹⁷<http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Presentacion-XP.pdf>

2.1.6. RESPONSABILIDADES DE CLIENTES Y DESARROLADORES EN XP

2.1.6.1. Responsabilidades de los Clientes

Los clientes son responsables de:

- Escribir *userstories* efectivas, que definan funcionalidad de tamaño apropiado, independientes, generen valor para el usuario o para él mismo (es responsable de que el producto tenga algún sentido) y de que sean testeables.

2.1.6.2. Responsabilidades de los Desarrolladores

Los desarrolladores son responsables de:

- Ayudar al cliente a escribir *userstories* efectivas, y conseguir toda la información necesaria mediante la conversación y quizá algunas de las técnicas que describan por ejemplo: Prototipos, Demos o *Storyboards*.
- Por ejemplo si se desea incluir una *story* sobre el uso de una tecnología o una pieza de infraestructura, se debe intentar describir la necesidad en términos de valor para el usuario o el cliente.

2.1.7. ALGUNOS ASPECTOS CULTURALES DE XP

XP es una cultura verbal en lugar de escrita.

- El código comunica el mensaje
- La documentación no es necesaria excepto cuando el código no puede comunicar el mensaje.

Se emplean métodos de lectura sin documentación.

- Juego de pruebas unitarias
- “Simplemente pregunta”

Se supone que el lector es conversador en la cultura

- Se basa en un lenguaje y valores compartidos entre los miembros del equipo.

Son lenguajes y valores diferentes de los de “Programadores de Producto” donde:

- La documentación se emplea para comunicar valores y lenguaje.
- Participantes externos, con diferentes valores.

2.1.8. CARACTERÍSTICAS DE XP¹⁸

XP está orientado a las personas y no a los procesos, existe una retroalimentación continua entre cliente y desarrollador. El cliente o el usuario se convierte en miembro del mismo equipo.

La metodología XP puede ser utilizada para proyectos pequeños, el equipo de trabajo puede ser de 2 a 20 personas. Se basa en poner junto a todo el equipo y emplear prácticas sencillas, con suficiente realimentación para facilitar que el equipo vea donde está y ajuste las prácticas a su situación concreta.

XP propone utilizar lenguajes de programación modernos, los cuales su tecnología este orientada a objetos.

Reduce el costo del cambio en todas las etapas del ciclo de vida del sistema.

Combina las mejores prácticas para desarrollar software y las lleva al extremo.

La metodología XP fue creada a base de prueba y error.

¹⁸<http://procesosdesoftware.wikispaces.com/METODOLOGIA+XP>

Tiene mayor énfasis en el desarrollo del software más que una buena documentación, además éste empieza en pequeño y de acuerdo al avance del proyecto se añade más funcionalidad con retroalimentación continua. Esta funcionalidad no es introducida antes de que sea necesaria.

XP no requiere herramientas que estén fuera de la programación y prueba. Además utiliza la comunicación oral tanto para el diseño como para los requerimientos, a diferencia de otras metodologías que usan el modelado. Beck [Beck99b] ha llegado a decir: “también yo creo en el modelado; sólo que lo llamo por su nombre propio; ‘mentir’, y trato de convertirlo en arte”.

2.1.9. ARTEFACTOS DE LA METODOLOGÍA XP¹⁹

Entre los artefactos de la metodología XP encontramos:

- Historias de Usuario.
- Tareas de Ingeniería.
- Pruebas de Aceptación.
- Tarjetas CRC.

2.1.9.1. LAS HISTORIAS DE USUARIO O USER STORIES²⁰

Describen la funcionalidad de un sistema de software que aporta valor al usuario y/o al cliente.

Es una técnica utilizada para especificar los requisitos del software. Son tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe tener, ya

¹⁹<http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Presentacion-XP.pdf>

²⁰http://elblogdelfrasco.blogspot.com/2008_07_01_archive.html

sean éstos requisitos funcionales o no funcionales. Cada historia de usuario es lo suficientemente comprensible y delimitada para que el programador la pueda implementar en unas semanas, además estas tarjetas son dinámicas y flexibles de tratar.

Se debe evitar detalles de especificación tecnológica, capa de base de datos y algoritmos.

Se debe tratar de que las historias estén enfocadas a las necesidades y beneficios del usuario.

Las historias de usuario son un artefacto de requerimientos de muy alto nivel.

Las historias de usuario deben tener:

- Una descripción escrita de la funcionalidad, usada para planificar y como recordatorio (se escriben en post-it y se pegan en pizarras para que el equipo las tenga presentes mientras están desarrollando).
- Conversaciones sobre la historia, las cuales sirven para despejar las dudas y los detalles de la funcionalidad.
- Pruebas de aceptación, que sirven como documentación y pueden ser usadas para determinar cuándo una historia está completa y libre de bugs.

Tomando en cuenta estos puntos, Beck presenta un ejemplo de ficha (customerstory and taskcard), en la cual se puede encontrar los siguientes contenidos:

- Fecha.
- Tipo de actividad (nueva, corrección, mejora).
- Prueba funcional.
- Número de historia.
- Prioridad técnica y del cliente.
- Referencia a otra historia previa.

- Riesgo.
- Estimación técnica.
- Descripción.
- Notas.
- Lista de seguimientos con la fecha.
- Estado
- Cosas por terminar.
- Comentarios

Historia de Usuario	
Número: 3	Usuario:
Nombre historia:	
Prioridad en negocio:	Riesgo en desarrollo:
Puntos estimados:	Iteración asignada:
Programador responsable:	
Descripción:	
Observaciones:	

Figura.12 Historia de Usuario

En la planificación, las historias pueden durar de una a tres semanas (para no superar el tamaño de una iteración). Las historias de usuario se descomponen en tareas de

programación y se asignan a los programadores para ser implementadas durante una iteración.

Las *Historias de usuario* son la unidad más pequeña de incremento del sistema y la unidad de estimación y control en un entorno ágil. Incluye los objetivos y motivaciones del usuario.

Algunas consideraciones importantes al escribir las historias de usuario:

Stakeholders escriben las historias del usuario.

Los stakeholders del proyecto deben escribir las historias de usuario, no los diseñadores.

Las historias de usuario son bastante simples, que las personas pueden aprender a escribirlas en pocos minutos, no tiene sentido que los expertos las escriban.

Use la herramienta más simple.

Las historias del usuario son escritas en tarjetas de índice. Estas tarjetas son muy fáciles trabajar y son por consiguiente una técnica modelada inclusiva.

Recuerde los requerimientos no-funcionales.

Las historias pueden describir una gran variedad de tipos de requerimientos.

Por ejemplo: *Los estudiantes pueden matricularse online*, esta historia de usuario es un requisito de uso similar a un caso de uso. *Considerando que las transcripciones serán vía online en un navegador normal* es más parecido a un requisito técnico.

Indique el tamaño estimado.

Se debe incluir una estimación del esfuerzo en llevar a cabo la historia del usuario. Una manera de estimar es asignar puntos a cada tarjeta, una indicación relativa de cuánto tiempo tomará que un par de programadores lleven a cabo la historia. El equipo sabe entonces que si actualmente toma 2.5 horas por punto; para una historia que tenga una estimación de 4, tardará alrededor de 10 horas para llevarla a cabo.

Indique la prioridad.

Los requisitos, incluso defectos identificados como parte de las actividades de comprobación paralelas independientes o por el esfuerzo de operaciones y soporte, son priorizados por el stakeholders de su proyecto (o representantes del proyecto como los dueños) y agregadas a la pila en el lugar apropiado. Se puede mantener fácilmente una pila de requisitos priorizados moviendo las tarjetas apropiadamente alrededor. Se puede ver que la tarjeta de historia de usuario incluye una indicación de la prioridad; Se puede usar una escala 1 a 10, siendo 1 la prioridad más alta. Otras priorizaciones son posibles, a menudo se usan prioridades de High/Medium/Low en lugar de números y algunas personas asignarán a cada tarjeta un único número de orden de prioridad (por ejemplo 344, 345,...). Se debe escoger la mejor estrategia que trabaje bien para el equipo. Si la prioridad cambio algún punto en el pasado, es importante que el equipo pueda mover la tarjeta a otro punto en la pila. La estrategia de priorización debe soportar esta clase de actividades.

Incluya un único identificador.

Las tarjetas de historias de usuario tiene un único identificador, por ejemplo 173. La razón para hacer esto, es que si necesita podría mantener alguna clase de traceability entre la historia de usuario y otros artefactos, en particular las pruebas de aceptación.

2.1.9.1.1. DETALLANDO UNA HISTORIA DE USUARIO²¹

Una historia de usuario contiene una pequeña información de lo que necesitará hacer en el proyecto. Existen tres tiempos en común cuando hacer esto.

Durante análisis/modelo de JIT que ataca con los stakeholders.

Para obtener una historia de usuario debe haber una conversación con él, en la conversación explorará los detalles detrás de la historia. Es común crear con los usuarios los bocetos de las pantallas para así saber qué es lo que quiere. Así es fácil identificar el criterio de aceptación o confirmación, que usarán para validar la historia que ha sido implementada correctamente.

Entonces en la parte de atrás de la tarjeta de historia de usuario se escribirán las confirmaciones del usuario, como se muestra en la fig. Por supuesto se pueden utilizar otras herramientas más sofisticadas que las tarjetas para éste propósito.

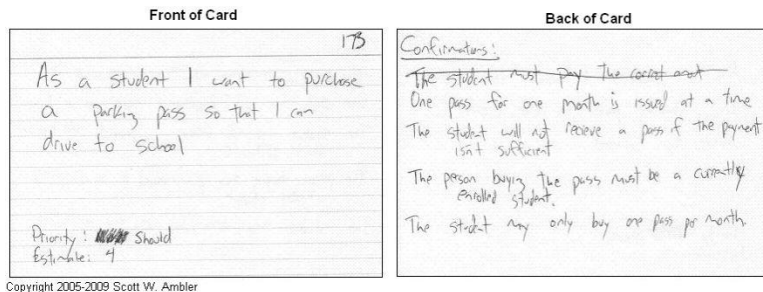


Figura.13 Historia de Usuario durante el análisis

²¹http://www.lsi.us.es/~javierj/investigacion_ficheros/PSISEXTREMA.pdf

Durante la planificación de la iteración.

Como parte del esfuerzo de estimación es bastante común listar tareas de la programación exigidas llevar a cabo en la historia de usuario.

Durante la aplicación.

Cuando se empieza a trabajar en la historia de usuario se puede decidir crear algunos bocetos ásperos de lo que se quiere construir, tal vez un mapa de flujo o UML actividad diagrama que representa la lógica comercial pertinente.

A continuación se describe brevemente el contenido de las historias de usuario.

2.1.9.1.2. DESCRIPCIÓN ESCRITA

La descripción escrita es sólo una parte muy pequeña de una historia de usuario. La parte más importante es la conversación, donde el usuario explica al desarrollador qué es exactamente lo que quiere y los detalles de la funcionalidad.

El desarrollador puede escribir anotaciones en la parte trasera del *post-it*, pero si lo hace tiene que ser una pequeña anotación de pocas palabras.

La forma de redactar la descripción escrita para *userstory* es:

Como <rol de usuario>, quiero <función del sistema> para poder <valor de negocio>

Es importante que en la descripción conste el **para qué**, esto permite tomar decisiones con respecto al diseño y de cómo resolver el problema.

Si el usuario no explica para qué quiere esa funcionalidad, es probable que el desarrollador no entienda toda la perspectiva del requerimiento y no codifique lo que el usuario necesita.

El rol del usuario debe estar bien especificado. Un error muy común es poner un usuario muy genérico o muy específico. **Cuando se escriben las *userstories* se habla de roles de usuario.**

Entonces un ejemplo de de la descripción escrita será la siguiente:

“Como estudiante, quiero matricularme vía oline para poder ingresar a primer nivel”

Para tener una idea más clara de cómo debe estar redactada una historia de usuario observe la fig.

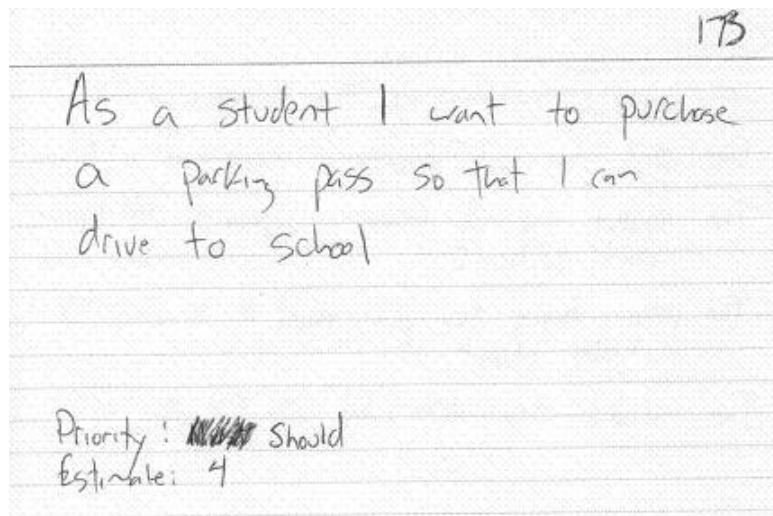


Figura.14 Historia de Usuario. Descripción escrita.

2.1.9.1.3. LAS HISTORIAS DEL USUARIO Y PLANEAMIENTO

Hay dos áreas dónde las historias del usuario afectan el proceso de la planificación cuando se está programando y cuando se está estimando. A continuación se explican cada una de estas áreas.

Programando.

Se describe el cambio ágil de dirección del proceso dónde trabaja los items, incluso las historias, son dirigidos en orden de prioridad. Así que, implica que la prioridad es asignada a una historia, la cual se afecta cuando se lleve a cabo la implementación del requisito. Ya se dijo que los clientes son los encargados de dar la prioridad a los requisitos. Además pueden repriorizar los requisitos existentes, al igual que definir nuevos requerimientos y añadirlos en la pila. También deben ser responsables para tomar las decisiones y proporcionar la información de una manera oportuna.

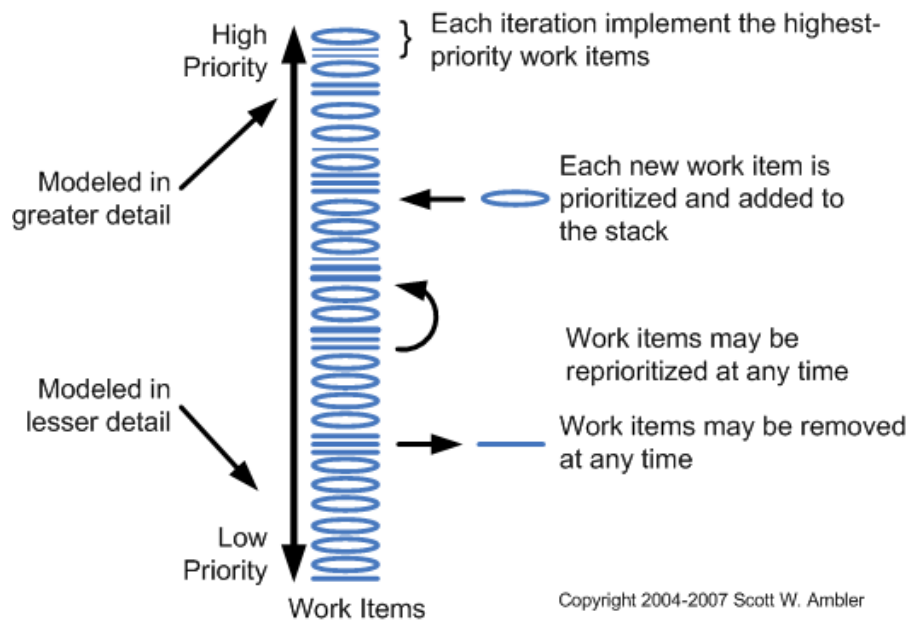


Figura.15 Historia de Usuario. Planeamiento Programado.

Estimando.

Los diseñadores son responsables de estimar el esfuerzo exigido para llevar a cabo una tarea, incluyendo las historias. Un diseñador puede realizar mucho trabajo en una iteración, el tamaño de los items de trabajo (incluso las historias), se ven afectados cuando son direccionados. Se puede creer que los diseñadores no tienen la habilidad para estimar los

requerimientos, por lo general esto sucede al inicio, el hecho es que no toma mucho tiempo para que las personas puedan hacer buenas estimaciones, ya que saben que tendrán que mantener estas estimaciones. Al utilizar la práctica de programación en parejas entonces una historia de usuario debe poder ser llevada a cabo por las dos personas en una sola iteración/velocidad. Si se trabaja en una iteración semanal cada historia de usuario debe describir menos valor en una semana de trabajo. Si no se trabaja en parejas de programadores las historias de usuario se aplicarán por una sola persona en una iteración. Grandes historias, son llamadas épicas, necesitarán dividirse en historias más pequeñas para encontrarse con este criterio.

2.1.9.1.4. DIVIDIR UNA USER STORY ÉPICA

Cuando una *userstory* es demasiado grande se dice que es **épica**.

Programar una **userstory** épica podría llevar mucho tiempo como un mes, un año, o más; la cuestión es que seguro va a tomar más tiempo que la duración de una iteración y va a resultar imposible estimarla. Por eso las *userstory* épicas hay que dividir las.

Para dividir las se puede considerar lo siguiente:

Por Datos:

Por ejemplo, en un formulario web se puede empezar con pocos datos. Se puede tener los campos: *Nombre del Postulante, Teléfono y Dirección*. Y más adelante agregar otra *userstory* con los campos que faltan.

Por Casos Especiales:

Si tengo un caso especial de que se tiene que registrar un extranjero, pero necesita llenar unos datos especiales y hay que dar de alta otros registros. El caso de alta de los extranjeros puede ir para otra *userstory*.

Por Operaciones:

Se podría dividir la Alta, Baja y Modificación en tres *userstories* diferentes.

Por Temas cross y no funcionales:

Podría empezar con una implementación que no maneje seguridad, log, manejo de errores, performance, volumen, e irlos agregando en siguientes iteraciones.

Por Prioridad:

No es bueno tener una *userstory* que abarca funcionalidad de distintas prioridades. Por ejemplo: "*Como vendedor de CtaCte quiero poder dar de alta y buscar facturas para poder almacenarlas y después imprimirlas como indica la ley*". Probablemente el dar de alta sea imprescindible, pero el buscar se pueda dejar para otra iteración.

Cuando las prioridades están mezcladas, se termina invirtiendo tiempo en funcionalidades de baja prioridad, dejando sin hacer otras de mayor prioridad.

Al dividir una *userstory* hay que tener en cuenta cuáles son los ***Minimum Marketable Features*** del producto. Esto es: el conjunto de mínimas funcionalidades con la que el producto puede instalarse en producción.

2.1.9.1.5. USER STORIES EFECTIVAS

Así como para UML tenemos buenas prácticas para escribir Casos de Uso Efectivos, también existen para *UserStories* y están descritas en detalle en el libro de **Mike Cohn**: "*UserStoriesAppliedFor Agile Software Development*".

Cohn dice que una buena *UserStory* debe ser:

Independiente:

No debe depender de otra *userstory*.

Negociable:

Las *userstories* no son obligaciones contractuales, simplemente son descripciones de funcionalidades, que pueden cambiar o negociarse en cualquier momento.

Valiosa para el Cliente o el Usuario:

Consideremos una *userstory* de este estilo: "*El software debe estar escrito en .NET*" o "*El programa se conectará a la Base de Datos a través de un pool de conexiones*". Al usuario realmente no le interesan estos detalles. Éstas no son *userstories*.

Estimable:

La funcionalidad que describe debe tener un tamaño que un desarrollador pueda estimar. Hay tres razones comunes por las que una *userstory* puede resultar imposible de estimar:

- 1) Falta de conocimiento del dominio por parte de los desarrolladores.
- 2) Falta de conocimiento técnico por parte de los desarrolladores.
- 3) La *userstory* es demasiado grande.

Pequeña:

Por todo lo anteriormente mencionado.

Testeable:

Deben poder ser testeadas para que los desarrolladores puedan asegurar que la funcionalidad está completa y funcionando correctamente.

Ejemplos de *userstories* no testeables:

1. *"Como operario quiero que el software sea fácil de usar..."*.
2. *"Como administrador quiero nunca tener que esperar demasiado para que una pantalla aparezca..."*.

¿Qué es fácil de usar? ¿Qué es esperar mucho?

2.1.9.2. TAREAS DE INGENIERÍA²²

Las tareas de ingeniería describen las actividades que se realizarán en el proceso descrito en una historia de usuario.

- Está relacionada con el número de historia.
- Se debe poner el nombre de la tarea.
- Los puntos estimados se pondrán dependiendo de lo q se crea se llevará en realizar esta tarea.
- Fecha de inicio y fin de la tarea.
- El nombre del programador responsable.
- Descripción de los puntos a tomar en cuenta para realizar la tarea

²²<http://users.dsic.upv.es/asignaturas/facultad/lsi/ejemploxp/index.html>

Tarea de Ingeniería	
Número Tarea:	Historia de Usuario (Nro. y Nombre):
Nombre Tarea:	
Tipo de Tarea : Desarrollo / Corrección / Mejora / Otra (especificar)	Puntos Estimados:
Fecha Inicio:	Fecha Fin:
Programador Responsable:	
Descripción:	

Figura.16 Tareas de Ingeniería

2.1.9.3. PRUEBAS DE ACEPTACIÓN

En una userstory es importante que quede bien definido cómo se va a aceptar, para esto las Pruebas de Aceptación automatizadas son ideales.

Deben ser automatizadas porque deben “abrazar al cambio” (lema fundamental del manifiesto ágil), se debe reaccionar de forma rápida ante él.

En un negocio donde los cambios se dan constantemente y el código es refactorizado todos los días, las pruebas de regresión se multiplican exponencialmente. Para un tester sería tediosa y muy susceptible a errores la tarea de probar lo mismo todos los días de forma manual.

Cuando un miembro del equipo está conversando con el usuario sobre una *userstory*, un tipo de anotación que puede hacer en la cara posterior del *post-it* es el de recordatorio de cómo testear la *story*. Por ejemplo, se podría escribir:

- *Probar subir un curriculum dañado*

- *Probar subir un curriculum vacío*
- *Probar dejar vacío el campo de Nombre*
- *Probar dejar vacío el campo de Remuneración Pretendida*

El cliente a menudo tiene muchas dificultades al escribir la funcionalidad de las pruebas, como desarrollador debe escribir las pruebas unitarias.

Para obtener una prueba de aceptación se debe seguir los siguientes pasos:

1. Identificar todas las acciones en la historia.
2. Por cada acción escribir dos pruebas.
3. Para algunos datos, suministrar las entradas que debería tener éxito, y llenar cualquier resultado satisfactorio.
4. Para otros datos, suministrar las entradas que tengan una acción fallida, y llenar la respuesta que debería tener.

Una forma de redactar la prueba de aceptación es en tres columnas: la acción a probar, los datos de prueba y el resultado esperado. A continuación se muestra un ejemplo:

<i>Action</i>	<i>Data</i>	<i>Expected Result</i>
1. Login	(Somedata) ValidUser / ValidPassword	Success: access to patient manager.
2.	(Otherdata) InvalidUser / InvalidPassword	Failure: login again

Figura.17 Plantilla de prueba de aceptación.

Generalmente no se obtiene toda la funcionalidad de la prueba, ya que no se responden todas las preguntas como:

- ¿Qué sucede cuando el nombre es válido y clave inválida? ¿Obtenemos el mismo resultado que si el nombre es inválido y la clave válida o si ambos son inválidos?
- ¿Cuántas veces se puede intentar acceder al sistema? ¿Infinitas veces?

Todas estas respuestas no están descritas en las historias de usuario, es el cliente quien conoce todas estas respuestas. Las pruebas de aceptación no están basadas en las historias de usuario sino en la funcionalidad que conoce el cliente asociada a dicha historia. Por tanto cualquier propuesta de generación de pruebas basada solo en historias de usuario será incompleta y, por eso, necesitamos que sea el usuario quien las escriba.

Una de las misiones del cliente es escribir estas pruebas, realizarlas sobre el sistema y dar su aprobación al mismo.

Entonces dado que el cliente es quien debe escribir las pruebas de aceptación, se muestra una guía de tal manera que sea sencilla y la puede escribir cualquier persona, es decir que no necesita tener ningún conocimiento en ingeniería de software.

Los pasos a seguir son:

Descripción	Resultado
Identificar todos los posibles resultados observables de la historia	Listado de resultados observables.
Identificar los resultados que terminan la historia y los que permiten continuar dentro la historia.	Listado de resultados observables clasificados en terminales y no terminales.
Identificar todos los caminos de ejecución posibles.	Listado de caminos de ejecución posibles y a cuál de los resultados identificados conduce.
Asignar un conjunto de valores válidos y valores del entorno a cada camino de ejecución para obtener el resultado esperado	Listado de caminos de ejecución, con sus resultados esperados y los valores que permiten obtener dicho resultado.
Eliminación de caminos redundantes.	Listado de caminos de ejecución, valores de prueba y resultados que se convertirán en pruebas de aceptación

Tabla. 3 Pasos para escribir una prueba de aceptación.

Ahora bien se da una explicación más clara de los pasos anteriores.

En el primer punto, el cliente debe ser capaz de enumerar y describir brevemente que consecuencias va a tener su historia de usuario.

Un resultado observable puede ser una pantalla del sistema, un nuevo elemento almacenado / modificado o eliminado de una base de datos, un mensaje o petición que recibe otro ordenador o un servidor, etc. Es decir, algo que se pueda comprobar bien manualmente, bien mediante código.

Estos resultados pueden terminar o no la historia. Por ejemplo:

En una historia de inserción de clientes se puede estar insertando clientes hasta que pulsemos la opción de salir del formulario de inserción. Al insertar un cliente el resultado observable puede ser un mensaje en el mismo formulario de inserción indicando que se ha insertado correctamente.

El segundo punto se puede realizar durante el paso anterior, o posteriormente una vez que se tiene la lista de todos los resultados observables. En el ejemplo de la inserción de clientes, un mensaje en el formulario de inserción indicando que el cliente se almacenó correctamente es un resultado observable que permite continuar con la historia (permite seguir insertando clientes), mientras que pulsar la opción salir nos llevará a otro formulario (resultado observable) y pondrá fin a la historia (ya no podemos seguir insertando clientes).

Un camino de ejecución es una descripción de las interacciones entre un actor o actores y el sistema, dentro de la historia de usuario, para llegar a uno de los resultados identificados en el punto 1. En el ejemplo de la inserción de clientes un posible camino de ejecución sería:

"insertar valores válidos en todos los campos del formulario y pulsar el botón de Insertar".

El resultado será, como mencionamos en el punto 1, volver al formulario con un mensaje que nos indique que el cliente se insertó correctamente.

A la hora de identificar los caminos de ejecución hemos de comprobar que cada resultado tenga, al menos un único camino de ejecución que conduzca a él. Si existen resultados para los que es imposible encontrar un camino de ejecución que nos conduzcan a ellos es posible que el cliente no haya definido la historia de usuario claramente o la historia abarca demasiada funcionalidad.

Es posible tener varios caminos de ejecución diferentes, e incluso infinitos caminos, con el mismo resultado. También, en este punto, podemos encontrar un camino de ejecución con varios resultados distintos, por ejemplo el camino de ejecución anterior podría terminar con un mensaje de error si el servidor de bases de datos no estuviera disponible.

Una vez identificados los conjuntos de pruebas para cada camino, evaluaremos si todos son necesarios o podemos eliminar algunos.

Es posible emplear sencillas notaciones gráficas como grafos de ejecución para simplificar la manera de expresar los caminos de ejecución.

Un valor válido es un valor que, aplicado a un camino de ejecución, nos permite obtener el resultado esperado. Un valor del entorno es algo que no depende del sistema, sino que es externo a él. En el ejemplo de la inserción de clientes, un valor del entorno será el servidor de bases de datos. Según su valor, disponible o no disponible, se debe obtener un resultado observable diferente a la hora de insertar un cliente.

Se puede encontrar caminos de ejecución redundante cuando se tenga caminos de ejecución iguales, o un camino contenido en otro, que, para los mismos valores genere los mismos resultados. También se puede encontrar caminos de ejecución infinitos especialmente en aquellos que no terminan la historia.

Por ejemplo, en la inserción de clientes sería posible insertar clientes correctos infinitamente.

En la figura se puede observar el formato que debe tener una prueba de aceptación.

Caso de Prueba de Aceptación	
Código:	Historia de Usuario (Nro. y Nombre):
Nombre:	
Descripción:	
Condiciones de Ejecución:	
Entrada / Pasos de ejecución:	
Resultado Esperado:	
Evaluación de la Prueba:	

Figura.18 Pruebas de Aceptación

2.1.9.4. TARJETAS CRC²³

Cada tarjeta representa una clase en la programación orientada a objetos y define sus responsabilidades (lo que ha de hacer) y la colaboración con las otras clases (cómo se comunica con ellas).

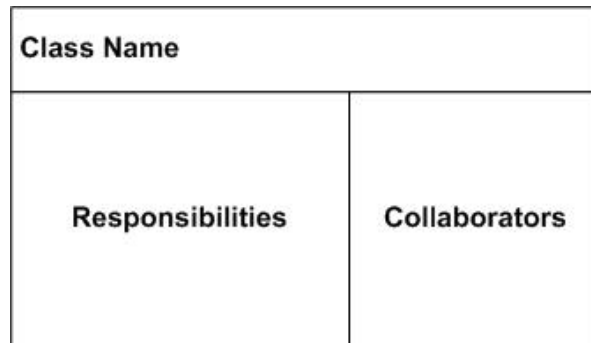


Figura.19 Tarjeta CRC.

Las responsabilidades son las cosas que se sabe las clases van a hacer. Es importante saber que una clase puede cambiar los valores de las cosas conocidas, pero es incapaz de cambiar los valores de otras clases conocidas.

Por ejemplo en la fig. se muestra que se sabe que tiene un estudiante: nombre, dirección, teléfono, éstas son las cosas que se conocen del estudiante. Ahora bien se sabe que el estudiante puede enrolarse en el Seminario, borrar un seminario y buscar transcripción, estas son las cosas que hace el estudiante, las cosas conocidas de la clase y todas ellas constituyen sus responsabilidades.

²³<http://www.ogis-ri.co.jp/otc/swec/process/am-res/am/artifacts/crcModel.html>

Student	
Student number Name Address Phone number Enroll in a seminar Drop a seminar Request transcripts	Seminar

Figura.20 Ejemplo de Tarjeta CRC

La colaboración toma uno de dos formas:

- Una solicitud de la información o
- Una solicitud para hacer algo.

Por ejemplo en la fig. Si la tarjeta de estudiante pregunta una indicación del la tarjeta de seminario si existe un espacio disponible, es una solicitud de información. Si el estudiante pide ser añadido al seminario, es una solicitud para hacer algo.

Otra forma seria que en el seminario se busque si existe un asiento libre entonces se matricule al estudiante de lo contrario enviar un mensaje al estudiante de que no ha sido inscrito en el seminario.

El valor más grande de tarjetas de CRC es permitir a las personas dejar el modo procedural y apreciar la tecnología del objeto. Permiten a todos los integrantes del grupo a contribuir en el plan, entre más quieran contribuir mejor será el plan.

Una tarjeta CRC puede ser una simulación del sistema, de cómo los objetos envían mensajes a otros objetos. Se colocan sobre un escritorio y se las hace rotar de tal manera que los miembros del grupo ven la mejor forma de cómo van a interactúa las clases.

Las tarjetas CRC son utilizadas como una estrategia a nivel de diseño. Son una herramienta efectiva de modelado conceptual para un diseño detallado. Lo excelente de diseñar con

estas tarjetas es que se puede limpiar el escritorio cuantas veces se quiera, y no se ha gastado gran cantidad de tiempo creando los diagramas para cada plan alternativo.

2.1.9.4.1. Cómo crear modelos CRC?

A continuación tenemos los siguientes pasos:

Encontrar las clases.-

Es una tarea fundamental porque se identifica los bloques de nuestra aplicación. Una buena regla es buscar de tres a cinco clases principales. Se pueden incluir otras clases inclusive las que representen a los actores, así se verá al estudiante en el mundo real (el actor) versus el estudiante en el sistema (la entidad).

Encontrar las responsabilidades.-

Aquí se debe preguntar qué es lo que hace una clase y así saber cuál es la información que se desea mantener de ésta.

Definir los colaboradores.-

Una clase a menudo no tiene la información suficiente para cumplir sus responsabilidades. Por consiguiente, debe colaborar (trabajo) con otras clases para conseguir hacer su trabajo. Para identificar los colaboradores para cada responsabilidad debe hacerse la pregunta. Tiene la habilidad para cumplir esta responsabilidad? Si no es así se busca una clase que tenga la habilidad de cumplir la funcionalidad perdida. Haciendo esto descubrirá la necesidad de nuevas responsabilidades en otras clases o a menudo puede necesitar una nueva clase o dos.

Mover las tarjetas alrededor.-

Todos entienden el sistema, las tarjetas deben ponerse en la mesa de manera inteligente. Dos tarjetas que colaboren con otra deben ponerse juntas. Considerando que las tarjetas que no colaboren con ella deben ponerse alejadas. Teniendo las tarjetas que colaboran entre si, es más fácil de entender las relaciones entre clases.

2.1.10. ROLES XP ²⁴

De acuerdo Beck tenemos los siguientes roles:

Programador.-

El programador escribe las pruebas unitarias y produce el código del sistema.

Es el responsable de decisiones técnicas y de construir el sistema.

Cliente.-

Es quien escribe las historias de usuario y las pruebas funcionales para validar la implementación del sistema.

Además asigna prioridades a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en apoyar mayor valor al negocio.

Encargado de pruebas (tester).-

Ayuda al cliente a escribir las pruebas funcionales.

Es el responsable de ejecutar las pruebas, difundir los resultados en el equipo y de las herramientas de soporte para las pruebas.

²⁴ www.programacionextrema.org

Encargado de seguimiento (tracker).-

Proporciona realimentación al equipo. Observa sin molestar. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones.

Realiza el seguimiento del progreso de cada iteración y conserva los datos históricos.

Entrenador (coach).-

Es el responsable del proceso global.

Proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.

Es el líder del equipo, toma las decisiones importantes.

Tiende a estar en un segundo plano a medida que el equipo madura.

Consultor.-

Es un miembro externo del equipo que posee conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.

Gestor (bigboss).-

Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

2.1.11. PROCESO XP: ²⁵

XP contempla los siguientes pasos:

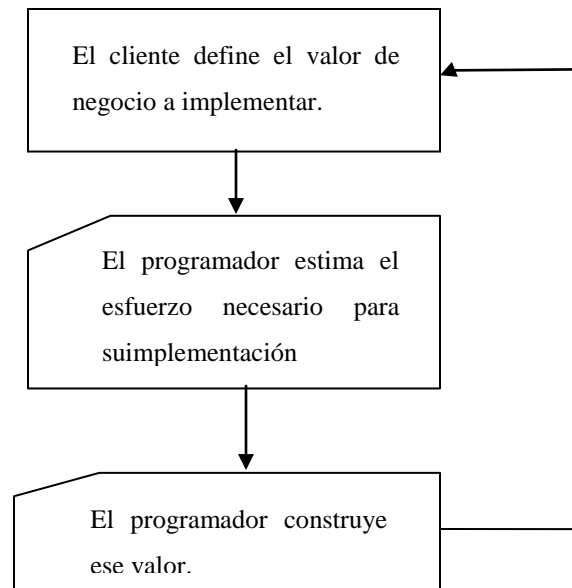


Figura.21 Proceso de XP

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software o no se cumplirán los plazos.

De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse de que el sistema tenga el mayor valor de negocio posible.

2.1.12. CICLO DE VIDA DE XP

El ciclo de vida ideal de XP consiste en 6 fases:

- Exploración.
- Planificación de la entrega.

²⁵ Critica MA.pdf

- Iteraciones.
- Producción.
- Mantenimiento y
- Muerte del proyecto.

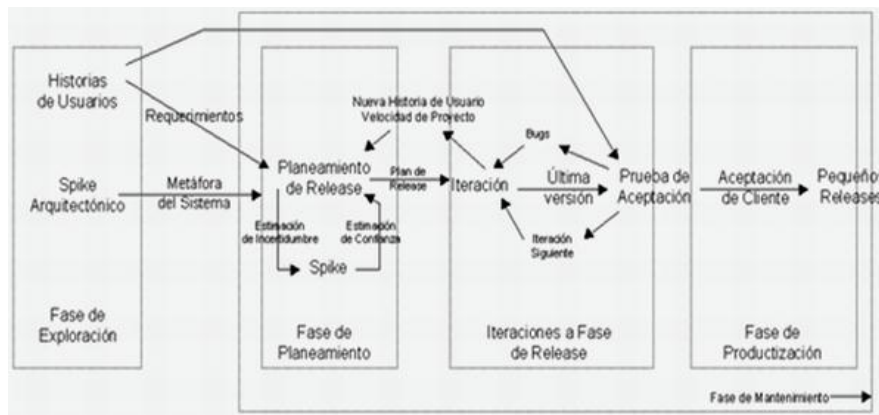


Figura.22 Ciclo de vida de XP²⁶

2.1.12.1. Exploración:

En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto.

El equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto.

Se construye un prototipo para probar la tecnología y explorar las posibilidades de la arquitectura del sistema.

La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

²⁶ <http://oness.sourceforge.net/proyecto/html/ch05s02.html>

2.1.12.2. Planeamiento:

Se priorizan las historias de usuario y se define el alcance del release.

Los programadores estiman cuánto esfuerzo requiere cada historia y a partir de allí se define el cronograma. La duración del cronograma del primer release no excede normalmente dos meses.

La fase de planeamiento toma un par de días. Para tener un release se deben incluir varias iteraciones.

El cronograma que se fija en esta etapa se realiza a un número de iteraciones, cada una toma de una a cuatro semanas en ejecución. La primera iteración crea un sistema con la arquitectura del sistema completo. Se logra esto, seleccionando las historias que harán cumplir la construcción de la estructura para el sistema completo.

El cliente decide las historias para cada iteración. Las pruebas funcionales se ejecutan al final de cada iteración. Al final de la última iteración el sistema está listo para producción.

2.1.12.3. Producción:

Se debe realizar pruebas y una comprobación extra del funcionamiento del sistema antes de poder entregar un release al cliente.

En esta fase, se pueden encontrar nuevos cambios y se debe tomar la decisión de incluir o no en el release actual.

Además, las iteraciones pueden ser aceleradas de una a tres semanas. Las ideas y las sugerencias postpuestas se documentan para una puesta en práctica posterior por ejemplo en la fase de mantenimiento.

Después de que se realice el primer release productivo para uso del cliente, el proyecto de Xp debe mantener el funcionamiento del sistema mientras que realiza nuevas iteraciones.

2.1.12.4. Mantenimiento:

Requiere de un mayor esfuerzo para satisfacer las tareas del cliente.

Cuando el sistema este en producción, la velocidad del desarrollo puede desacelerar.

Esta fase puede requerir la incorporación de nueva gente y cambiar la estructura del equipo.

2.1.12.5. Muerte:

Esta fase se cumple cuando el cliente no tiene más historias para ser incluidas en el sistema.

Requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema.

Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura.

La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo.

2.1.13. VALORES DE LA PROGRAMACIÓN EXTREMA

Los principios originales de la programación extrema son:

- Simplicidad:
- Comunicación:
- Retroalimentación (feedback):
- Coraje o valentía.

Un quinto principio, el respeto, fue añadido en la segunda edición de *Extreme Programming Explained*.

A continuación detallaremos los principios.

2.1.13.1. SIMPLICIDAD

Es la base de la programación extrema. El diseño se simplifica para agilizar el desarrollo y facilitar el mantenimiento. Un diseño complejo del código junto a sucesivas modificaciones realizadas por diferentes desarrolladores hacen que la complejidad aumente exponencialmente.

La refactorización del código es importante para mantener la simplicidad, es decir que el código se mantenga simple a medida que crece.

La simplicidad también se aplica en la documentación, así el código debe comentarse en su justa medida, pero siempre se debe intentar autocommentar el código. Para conseguir esto se deben elegir nombres adecuados para las variables, métodos y clases. Los nombres largos no decrementan la eficiencia del código ni el tiempo de desarrollo ya que actualmente existen herramientas de autocompletado y refactorización.

La simplicidad junto con la autoría colectiva del código y la programación por parejas se asegura que cuanto más grande se haga el proyecto, todo el equipo conocerá más y mejor el sistema completo.

2.1.13.2. COMUNICACIÓN

La comunicación se realiza de diferentes formas.

Para los programadores el código comunica mejor cuanto más simple sea. Si el código es complejo hay que esforzarse para entenderlo.

El código autodocumentado es más fiable que los comentarios, ya que en ocasiones pueden quedar desfasados con el código a medida que es modificado. Se debe comentar sólo lo que no va a variar, por ejemplo el objetivo de una clase o la funcionalidad de un método.

Las pruebas unitarias son otra forma de comunicación ya que describen el diseño de las clases y los métodos al mostrar ejemplos concretos de cómo utilizar su funcionalidad. La programación en parejas es otra forma de comunicarse ya que los programadores están constantemente comunicándose. El cliente al formar parte del equipo de desarrollo la comunicación es más fluida, además el cliente decide que características tienen prioridad y siempre debe estar disponible para solucionar dudas.

2.1.13.3. REALIMENTACIÓN(FEEDBACK):

El cliente al estar integrado al equipo de trabajo, su opinión sobre el estado del proyecto se conoce en tiempo real, logrando así realimentar la funcionalidad que debe tener el sistema.

En XP al trabajar con ciclos muy cortos de los cuales se tienen resultados, se minimiza el tener que rehacer partes que no cumplen con los requisitos y ayuda a los programadores a centrarse en lo que es más importante.

En un proyecto en el cual tenga ciclos muy largos puede ocasionar muchos problemas, por ejemplo meses de trabajo pueden eliminarse debido a cambios en los criterios del cliente o malentendidos por parte del equipo de desarrollo.

El código también es una fuente de retroalimentación gracias a las herramientas de desarrollo. Por ejemplo, las pruebas unitarias informan sobre el estado de salud del código.

Ejecutar las pruebas unitarias frecuentemente permite descubrir fallos debidos a cambios recientes en el código.

2.1.13.4. CORAJE O VALENTÍA

Se preguntaría ¿por qué el coraje es un principio de XP? Para muchos gerentes o directivos empresas no es fácil aceptar que se aplique la programación en parejas ya que piensan que sólo la mitad de los programadores se encuentran trabajando y esto afectaría a la productividad. Entonces los gerentes deben ser valientes para confiar en que la programación en parejas no afecta en la productividad y que se obtendrá código de calidad.

Se requiere coraje para implementar las características que el cliente quiere ahora sin caer en la tentación de optar por un enfoque más flexible que permita futuras modificaciones.

No se debe emprender el desarrollo de grandes marcos de trabajo (frameworks) mientras el cliente espera. En ese tiempo el cliente no recibe noticias sobre los avances del proyecto y el equipo de desarrollo no recibe retroalimentación para saber si va en la dirección correcta.

La forma de construir marcos de trabajo es mediante la refactorización del código en sucesivas aproximaciones

2.1.13.5. RESPETO

El respeto se manifiesta de varias formas:

Los miembros del equipo se respetan los unos a otros, porque los programadores no pueden realizar cambios que hacen que las pruebas existentes fallen o que demore el trabajo de sus compañeros.

Además los miembros del equipo respetan su trabajo porque siempre están luchando por la alta calidad del producto y buscando el diseño más óptimo para la solución a través de la refactorización del código.

2.1.14. PRINCIPIOS BÁSICOS DE LA METODOLOGÍA XP

Los principios fundamentales se apoyan en los valores y son cuatro. Se busca:

1. Realimentación veloz,
2. Modificaciones incrementales,
3. Trabajo de calidad y
4. Asunción de simplicidad.

2.1.15. BUENAS PRÁCTICAS PARA LA APLICACIÓN EXITOSA DE LA METODOLOGÍA XP²⁷

La mayoría de las prácticas propuestas por XP no son nuevas sino que ya habían sido propuestas en ingeniería del software e incluso demostrado su valor en la práctica. El mérito de XP es integrarlas de una forma efectiva y complementarlas con otras ideas desde la perspectiva del negocio, los valores humanos y el trabajo en equipo.

XP propone las siguientes prácticas para ayudar en el desarrollo de software.

- El juego de la planificación
- Entregas pequeñas
- Metáfora.

²⁷ Xtremeprogramming.org

- Diseño simple
- Pruebas.
- Refactorización (refactoring)
- Programación en parejas
- Propiedad colectiva del código
- Integración continua
- 40 horas por semana
- Cliente in-situ
- Estándares de programación.

El juego de la planificación

Hay una comunicación frecuente entre el cliente y los programadores.

El cliente tendrá que escribir sus necesidades definiendo las actividades que realizará el sistema, para esto debe crear un documento llamado *Historias del usuario* (UserStories).

Dependiendo de la complejidad que tenga el sistema se deben definir de 20 a 80 historias, las cuales son suficientes para poder realizar el llamado Plan de Liberación. En el Plan de liberación se especifican los tiempos en los cuales se deben entregar los prototipos de la aplicación para luego recibir retroalimentación por parte del cliente. Además el Plan ayuda a los técnicos a tomar decisiones técnicas y a los clientes a tomar las decisiones del negocio

El equipo técnico realiza una estimación del esfuerzo que se requiere para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración, además determinan cuál historia es más importante y tiene mayor prioridad.

Generalmente, cada una de las historias de usuarios necesita de una a tres semanas para su desarrollo. El tiempo se debe determinar dependiendo de de la complejidad de cada historia, el que se crea conveniente para cumplir la historia de usuario.

Realizar reuniones periódicas es muy importante en esta fase. Las reuniones pueden ser diarias, en las cuales el equipo de desarrollo puede identificar problemas, proponer soluciones e indicar los puntos en los cuales se deben dar más importancia dependiendo de la dificultad.

Las historias del usuario están impresas o escritas en tarjetas. Entonces diseñadores y clientes mueven las tarjetas alrededor en una mesa grande, se crea un juego de historias, para saber cuál de ellas se debe realizar primero. Se puede hacer una planificación por tiempo o por alcance. El tiempo es determinado por cuántas historias de usuario pueden realizarse antes de una fecha dada. El alcance se refiere cuanto tiempo llevará terminar las historias.

Entregas pequeñas

Se pone en producción una versión del sistema que sea operativa, aunque no tenga toda la funcionalidad del sistema. Estas versiones se actualizan rápidamente y constantemente permitiendo evaluar en un ambiente real el verdadero valor del negocio ya que en cada entrega se obtiene un resultado. Las entregas van de 2 a 3 semanas.

Metáfora.

Una metáfora es una historia que define como funciona el sistema completo (conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema, ayudando a la nomenclatura de clases y métodos del sistema) con una frase o nombre, de tal manera que se tiene una idea de qué es lo que hace cada parte del programa. Un ejemplo claro es el "recolector de basura" de java. Ayuda a que todos los programadores (y el cliente) sepan de qué estamos hablando y que no haya mal entendidos.

Son desarrolladas por los programadores al inicio del proyecto. Las metáforas pueden ser compartidas por el cliente y el equipo de desarrollo.

XP propone que las historias sean descripciones breves del trabajo del sistema en lugar de los tradicionales diagramas y modelos UML. Es decir que las metáforas deben proporcionar una visión general y un entendimiento común del sistema y sus funciones, y de cómo se debe construir, define el alcance y propósito del sistema. Ayudan a conseguir una visión común y a transformar el conocimiento tácito en conocimiento explícito.

Diseño Simple.

Se basa en la filosofía de que el mayor valor de negocio es entregado por el programa más sencillo que cumpla los requerimientos. Es por esto que se debe diseñar una solución simple que pueda funcionar y ser implementada en un momento determinado del proyecto, el diseño debe cumplir con las necesidades inmediatas del cliente.

Este proceso permite eliminar redundancias y rejuvenecer diseños obsoletos, todo esto de forma sencilla.

Las tarjetas CRC (Clase, Responsabilidad y Colaboración) también ayudarán al equipo a definir actividades durante el diseño del sistema.

Al utilizar esta técnica es fácil reconocer los problemas y debilidades del proceso. Si están varias personas del grupo hablando y moviendo las tarjetas, entonces se debe limitar a que se encuentren solo dos personas. Cuando una persona se sienta la otra se pone de pie.

Una de las más grandes críticas que tiene las tarjetas CRC es la falta de un diseño escrito, ya que el diseño parece hacerse obvio. Una tarjeta puede escribirse por completo y puede tenerse como documentación.

Pruebas.

Las pruebas unitarias dirigen la producción de código. Son establecidas por el cliente y ejecutadas constantemente en cada modificación del sistema.

Combinar el tiempo para crear las pruebas de unidad y crear algo de código, será fácil.

Las pruebas unitarias ayudan al desarrollador a saber realmente que es lo que necesita hacer. Los requisitos son insertados firmemente bajo las pruebas. No puede haber mal entendidos en la especificación escritas en la forma de código fuente. Además se cuenta con la retroalimentación inmediata cuando se trabaja. A menudo no es clara cuando un desarrollador ha terminado la funcionalidad. En el alcance son consideradas las extensiones y las condiciones de error. En los sistemas primero se realiza el código y luego se realiza las pruebas y por lo general son realizados por equipos diferentes.

Creando las pruebas unitarias el diseño será influenciado por el deseo de probar todo el sistema al cliente.

Existe un ritmo para crear una prueba de unidad en un sistema en desarrollo, primero se crea la prueba para definir algún aspecto del problema, entonces luego se crea el código muy simple de tal manera que pase la prueba. Se realizarán las pruebas necesarias hasta que no exista más que probar y por ende el código será simple.

La unidad de prueba puede ayudar a formalizar los requisitos, clarificar la arquitectura, escribir el código, debug el código, integrar el código, liberar, optimizar y el curso de la prueba.

Durante la vida del proyecto una prueba automatizada puede salvar cien veces el costo de crearlo, encontrando y cuidando contra los bugs.

Un concepto erróneo es crear las pruebas unitarias en los últimos tres meses de desarrollo del proyecto. Las pruebas llevan tiempo para evolucionar, ya que no es fácil descubrir todos los problemas que pueden ocurrir. Cuando se crean estas pruebas se guarda la funcionalidad que puede dañarse accidentalmente.

La unidad de prueba habilita la refactorización, después de cada pequeño cambio en las pruebas puede verificar que cambie la estructura y no existan cambios en la funcionalidad. Arreglar un pequeño problema cada pocas horas toma menos tiempo que arreglar un gran problema antes de la fecha de entrega. Con las pruebas unitarias se puede fusionar los cambios en la última versión y tener una release en menos tiempo.

Las pruebas unitarias proveen una seguridad neta de las pruebas de regresión y las pruebas de validación que pueden refactorarse e integrarse efectivamente.

Si se quiere saber cómo va a ser el funcionamiento del sistema es recomendable utilizar ambientes de Prueba (Unittestingframeworks) en los cuales se pueden hacer varias simulaciones del sistema en funcionamiento.

Un ejemplo de un ambiente de prueba es el JUnit para Java.

Crear las pruebas unitarias antes de crear el código ayuda a solidificar los requisitos, mientras mejora el enfoque del desarrollador.

Además se establece un periodo de pruebas de aceptación del funcionamiento del sistema el cual es llamado periodo caja negra, en este periodo de pruebas se deben definir las entradas para el sistema y los resultados que esperamos obtener.

Las pruebas de aceptación son creadas desde las historias de usuario. El usuario especifica el escenario a probar cuando la historia ha sido implementada correctamente. Una historia de usuario puede tener una o varias pruebas de aceptación, esto asegura la funcionalidad del trabajo.

Cada prueba de aceptación representa algún resultado esperado del sistema. Los clientes son los responsables de verificar la exactitud de la prueba de aceptación

Al final de este proceso el cliente contará con una descripción clara de todas las pruebas de aceptación que debe generar para verificar completamente la funcionalidad resumida de una historia de usuario.

Refactorización (refactoring)

El equipo de desarrollo mejora el diseño del sistema a través de todo el proceso de desarrollo. La refactorización es una actividad constante, en ella se reestructura el código evitando así la duplicidad y/o ineficiencia, mejorando su legibilidad, simplificarlo y hacerlo más flexible para facilitar cambios futuros, manteniendo un sistema enfocado a proveer el valor del negocio. Es decir se mejora la estructura interna del código sin alterar el comportamiento externo.

Cuando se remueve redundancia, se elimina funcionalidad inutilizada y se rejuvenece diseños obsoletos, se está refactorizando.

Refactorizar durante todo el desarrollo del proyecto ahorra tiempo e incrementa la calidad. Además refactorizar implacablemente para guardar un diseño simple, evitará un desorden innecesario y la complejidad.

Se debe asegurar de que el código sea limpio y conciso así será fácil de entender, modificar y extender. Si el código es bien cuidado se tarda menos tiempo en producirlo.

Programación en parejas

Uno de los principios más radicales de XP es la programación en parejas y la mayoría de gerentes de desarrollo tiene sus dudas, ya que toda la producción de código se realizara de esta forma. Esto conlleva ventajas implícitas (menor tasa de errores, mejor diseño, mayor satisfacción de los programadores...).

Los programadores se ponen en parejas, compartirán una sola máquina. Cada miembro de la pareja juega su papel:

Uno codifica en el computador y piensa la mejor manera de hacerlo.

El otro piensa mas estratégicamente, ¿Va a funcionar?, ¿Puede haber pruebas donde no funcione?, ¿Hay forma de simplificar el sistema global para que el problema desaparezca?

La tarea de los dos programadores es codificar las historias que les corresponde, realizarán pruebas del funcionamiento una y otra vez hasta que lo programado pase con éxito todas las pruebas.

Generalmente las parejas de programadores se intercambian frecuentemente, entonces todos trabajan con todos. Por ejemplo en la mañana un programador tiene un compañero de trabajo y en la tarde puede trabajar con otro compañero.

El trabajo de programar en parejas que se encuentran constantemente intercambiándose tiene las siguientes ventajas:

- Cuatro ojos ven más que dos. Al trabajar de dos en dos, el código será de mayor calidad desde el mismo momento de crearlo y tendrá menos fallos.
- Los programadores novatos aprenderán de los expertos al emparejarse con ellos.
- Si una pareja realiza un trozo de código susceptible de ser reutilizado en el proyecto, hay dos programadores que lo saben y que lo reutilizarán cuando puedan (ya que saben cómo funciona), enseñándolo a sus nuevos compañeros. De esta manera el conocimiento del código ya hecho se propaga de forma natural entre todos los programadores del equipo.
- El estilo de programación tiende a unificarse.

Además hay que tomar en cuenta las siguientes reglas:

- No deben estar dos programadores que son novatos en la programación en parejas, siempre debe estar un programador experimentado y un programador recién llegado.
- Cuando se decida trabajar separadamente; pero con la responsabilidad de juntarse luego, esto realmente no es programación en parejas.
- Si los dos programadores no ven que es lo que sucede en su monitor, no están programando en pareja.
- Todos trabajan en parejas.
- Los programadores tienen que aprender a confiar unos en otros, tomará tiempo para construir la confianza en el grupo.

Cuando una pareja de programadores se dispone a realizar una historia de usuario, uno de ellos puede recordar haber realizado un trozo de código que puedan reutilizar, entonces la pareja lo reutilizará, lo modificará si es necesario. Si el código ha sido modificado deben

realizar las pruebas automáticas necesarias, asegurándose que los cambios realizados no han afectado el funcionamiento del código.

Si una pareja ve que un código es mejorable, lo mejoran, realizando luego las pruebas necesarias. Si ellos descubren que existe algún error en el código que las pruebas realizadas no lo detectan, crean nuevas pruebas que lo detecte y lo corrigen.

Como las parejas están formadas por un compañero con mucha más experiencia que el otro. Los primeros días de trabajo pueden parecer de enseñanza. El compañero de menor experiencia hace muchas preguntas y escribe muy poco en la computadora. Sin embargo, rápidamente, el menos experimentado empieza a encontrar los pequeños errores estúpidos, como los paréntesis desbalanceados. El más experimentado se da cuenta de la ayuda que recibe. Luego de unas pocas semanas más, el menos experimentado comienza a darse cuenta de los patrones generales que usa su compañero, y se da cuenta de las desviaciones de los mismos.

En un par de meses, típicamente, la distancia entre ambos compañeros ya no es tan notable como al principio. El menos experimentado escribe en la computadora más regularmente. La pareja nota que cada uno tiene sus fortalezas y debilidades. La productividad, calidad y satisfacción se elevan.

La programación en parejas no es una mala práctica de programación en ambientes malos. Un punto importante los efectos reprimidos es que la programación en pareja da lugar a programadores cowboy con una mentalidad JANGIT (JustAdd New GarbageIgnoringTests, Sólo Agregue Nueva Basura que Ignora las Pruebas). Si se tiene un codificador cowboy en el grupo, emparejarlo con un programador extremo experimentado virtualmente garantiza que el código será escrito apropiadamente y las pruebas necesarias se hagan.

XP proporciona cierto juego en el cual se hace checks y se equilibra un par de personas que no piensan de igual forma. Al combinar ésta con otras reglas de XP el resultado es la eliminación de las malas prácticas de programación y los malos ambientes. Programar en pareja se libera del problema del resultado de programadores mediocres que escriben código y documentación pobre.

Entonces se dirá que el trabajar en parejas producirá aplicaciones más buenas, de manera consistente, a iguales o menores costes.

Propiedad colectiva del código

Se refiere a un código con propiedad compartida. Todos son el propietario de todo. Cualquier programador puede cambiar cualquier parte del código en cualquier momento, arreglar los bugs o realizar refactorización. Ninguna persona se vuelve un cuello de botella para los cambios.

La propiedad colectiva del código anima a todos para contribuir con nuevas ideas en todos los segmentos del proyecto, arreglar los bugs o realizar refactorización.

Al principio será difícil de entender, es casi inaceptable que todo el equipo sea el responsable de la arquitectura del sistema, y no sea un solo arquitecto quien tenga una visión del sistema.

Pero por ejemplo si se le hace una pregunta al arquitecto, no es raro conseguir una respuesta que no sea la adecuada. Un sistema non-trivial no puede estar en la mente de una sola persona. Los programadores trabajarán duro por el cambio del sistema fuera de la visión del arquitecto. Entonces la arquitectura del sistema debe ser conocido por todo el equipo de trabajo logrando así que ellos también tengan un poco de responsabilidad en las decisiones arquitectónicas y no sólo recibir órdenes.

En la práctica la propiedad del código colectiva es realmente más fiable que poner a una sola persona a cargo de mirar las clases específicas. Especialmente si una persona puede dejar el proyecto en cualquier momento.

Es importante cambiar a los programadores a otra área de código, no sabe sólo de un área determinada. Todos los integrantes del grupo saben más del código de cada sección en lugar que una persona sepa todo el código de un área. El equipo se vuelve más flexible, el trabajo es igual para todos logrando así que el grupo sea más productivo.

Los defensores de XP argumentan que mientras haya más gente trabajando en un trozo de código, menos errores aparecerán.

Integración continua

Cada pieza de código, una vez que está lista es integrada en el sistema, permitiendo un rápido progreso implementando las nuevas características del software.

En lugar de crear builds (o versiones) estables de acuerdo a un cronograma establecido, el sistema puede llegar a ser integrado y construido varias veces en un mismo día. Esto reduce los problemas de integración comunes en proyectos largos y estilo cascada.

No deben hacerse los cambios al código obsoleto que causa dolores de cabeza al momento de la integración, es así que todos deben trabajar con la última versión.

La integración continua evita o revela los problemas de compatibilidad fácilmente. La integración es un "págume ahora o págume más tarde" la actividad. Es decir, si usted integra al proyecto en cantidades pequeñas, se encontrará que intenta integrar el sistema durante semanas hasta el fin del proyecto. Siempre se debe trabajar en el contexto de la última versión del sistema.

40 horas por semana

El equipo debe trabajar máximo 40 horas a la semana, Se debe trabajar tiempo extra cuando es necesario, pero no debe realizarse dos semanas seguidas. Si esto ocurre, probablemente existe un problema que debe corregirse.

Minimizar las horas extras y mantener los programadores frescos y descansados, evita la rotación de personal y generará código de mayor calidad. El trabajo extra desmotiva al equipo.

Cliente in-situ

El cliente debe estar disponible durante todo el desarrollo del proyecto, si el equipo lo requiere en cualquier tiempo él debe estar presente. Ya que el cliente es quien determina los requerimientos y define la funcionalidad del sistema y define las prioridades, él conduce al equipo de trabajo hacia lo que más aporte valor al negocio. Recordemos que el cliente es el usuario final del proyecto y por ende es quien mejor lo conoce.

El cliente no solo ayuda al equipo sino que es parte de éste.

Además al encontrarse siempre con el equipo de trabajo puede responder pronto las preguntas de los programadores y ellos podrán resolver inmediatamente los problemas asociados con sus dudas. Cualquier consulta de los requerimientos por pequeña que sea debe ser realizada directamente con él así como cualquier sugerencia o idea (esto es algo que a los clientes les gusta mucho). Esto evita el tener que adivinar lo que el cliente quiere y el tener que cambiar las cosas cuando el cliente ve lo que se ha hecho.

Además, el cliente obtiene una visibilidad completa de como se realiza el proceso de desarrollo y le permite comprender lo difícil que es el realizar software. Esto evita tener un cliente "gruñón" y este se transforma en un cliente comprensivo (literalmente).

La interacción cara a cara con el programador disminuye el tiempo de comunicación y la cantidad de documentación; ya que la comunicación oral es mejor que la escrita, así se logra reducir los altos costes de su creación y mantenimiento.

Estándares de programación.

Es importante que el código del proyecto mantenga ciertos estándares de programación para que sea legible, ya que los programadores se comunican a través de éste.

Los estándares definen la propiedad del código compartido así como las reglas para escribirlo y documentarlo. Las normas de codificación hacen que el código sea consistente y fácil de entender, así cualquier grupo de trabajo puede leerlo y refactorizarlo.

Al final lo que se consigue es que el código del sistema se vea como si un solo programador lo hubiera escrito.

Finalmente tenemos una representación de las prácticas de XP, se observa que todas ellas se apoyan y se refuerzan entre sí.

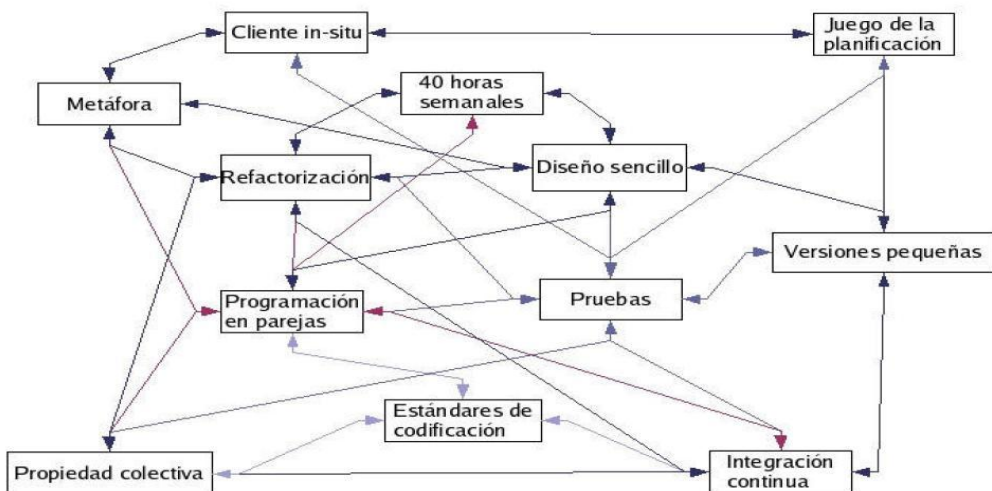


Figura.23 Prácticas de XP

2.1.16. ACTIVIDADES DE LA METODOLOGÍA XP

2.1.16.1. CODIFICAR

Según XP el código es lo más importante del desarrollo de un sistema, sin código no tiene nada.

La codificación puede ser dibujar diagramas que generarán código, hacer scripts de sistemas basados en web o codificar un programa que ha de ser compilado.

La codificación también puede usarse para entender la solución más apropiada.

Además puede ayudar también a comunicar pensamientos sobre problemas de programación. Un programador que trata con un problema de programación complejo y encuentra difícil explicar la solución al resto, puede codificarlo y usar el código para demostrar lo que quería decir.

El código, es siempre claro y conciso y no se puede interpretar de más de una forma. Otros programadores pueden dar retroalimentación de ese código codificando también sus pensamientos.

2.1.16.2. PROBAR

Nadie puede estar seguro de algo si no lo ha probado. Las pruebas no es una necesidad primaria percibida por el cliente. Mucho software se libera sin unas pruebas adecuadas y funciona.

XP dice que uno no puede estar seguro de que una función funciona si no la prueba. Esto sugiere la necesidad de definir de lo que uno puede no estar seguro.

- No puedes estar seguro de si lo que se ha codificado es lo que se quiere significar.

Para probar esta incertidumbre, XP usa pruebas unitarias. Estas son automatizadas y

prueban el código. El programador intentará escribir todas las pruebas que crea puedan cargarse el código que está escribiendo; si todas las pruebas se ejecutan satisfactoriamente entonces el código está completo.

- No se puede estar seguro de si lo que se quiere significar era lo que debería. Para probar esta incertidumbre, XP usa pruebas de aceptación basadas en los requisitos dados por el cliente.

2.1.16.3. ESCUCHAR

Los programadores no saben todo sobre el negocio del sistema bajo desarrollo. Para que los programadores encuentren cual debe ser la funcionalidad del sistema, deben escuchar las necesidades de los clientes. También tienen que intentar entender el problema del negocio y dar a los clientes retroalimentación, así se mejora el propio entendimiento del cliente.

2.1.16.4. DISEÑAR

Desde el punto de vista de la simplicidad, uno podría decir que el desarrollo de sistemas no necesita más que codificar, probar y escuchar. Si estas actividades se desarrollan bien, el resultado debería ser un sistema que funcionase. En la práctica, esto no ocurre. Uno puede seguir sin diseñar, pero un momento dado se va a atascar. El sistema se vuelve muy complejo y las dependencias dentro del sistema dejan de estar claras. Uno puede evitar esto creando una estructura de diseño que organice la lógica del diseño. Buenos diseños evitarán pérdidas de dependencias dentro de un sistema; esto significa que cambiar una parte del sistema no tendrá por qué afectar a otras.

2.1.17. FASES DE LA METODOLOGÍA XP²⁸

2.1.17.1. PLANIFICACIÓN

La planificación se elabora en etapas, donde se aplican diferentes iteraciones. Para conseguir esto es necesario plantear reglas las cuales deben ser seguidas por quienes estén involucrados en el proyecto.

En esta etapa se deben escribir las historias de usuario, las cuales son los casos de uso en terminología del cliente, así se puede obtener las pruebas de aceptación que permiten tener una estimación del tiempo de desarrollo.

Además se debe crear el plan de lanzamientos, el cual es un calendario que todos deben poder cumplir, para esto es importante que participen todas las personas involucradas en el proyecto, se toman como base de trabajo las historias de usuario.

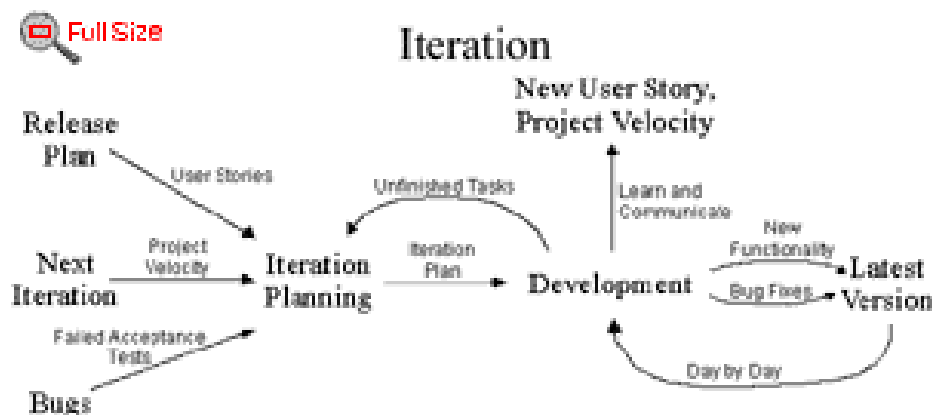


Figura.24 Iteraciones de XP

Las entregas se deben hacer lo más pronto posible, en cada iteración el cliente recibe una nueva versión. Son recomendables varias entregas y frecuentes, de esta forma se puede

²⁸http://synergyca.net/index.php?p=1_9_Metodologias

detectar errores rápidamente. En esta etapa son muy comunes los errores, por eso se establecen mecanismos de revisión. Las historias de usuario se deben revisar cada tres o cinco iteraciones y si es necesario renegociar la planificación.

Cada iteración necesita ser planificada, a esto se llama planificación iterativa, en la que se anotan las historias de usuarios que se desean y las que no han pasado las pruebas de aceptación. Esta planificación se puede hacer en tarjetas, en las que se escribirán los trabajos que durarán entre uno y tres días.

La planificación en iteraciones y el diseño iterativo, hace que aparezca una práctica que no existía en la programación tradicional. Se trata de las discusiones diarias informales, para fomentar la comunicación, y para hacer que los desarrolladores tengan tiempo de hablar de los problemas a los que se enfrentan y de ver cómo van con sus trabajos.

Además para evitar cuellos de botella y fomentar la propiedad colectiva del código, se deben cambiar de área.

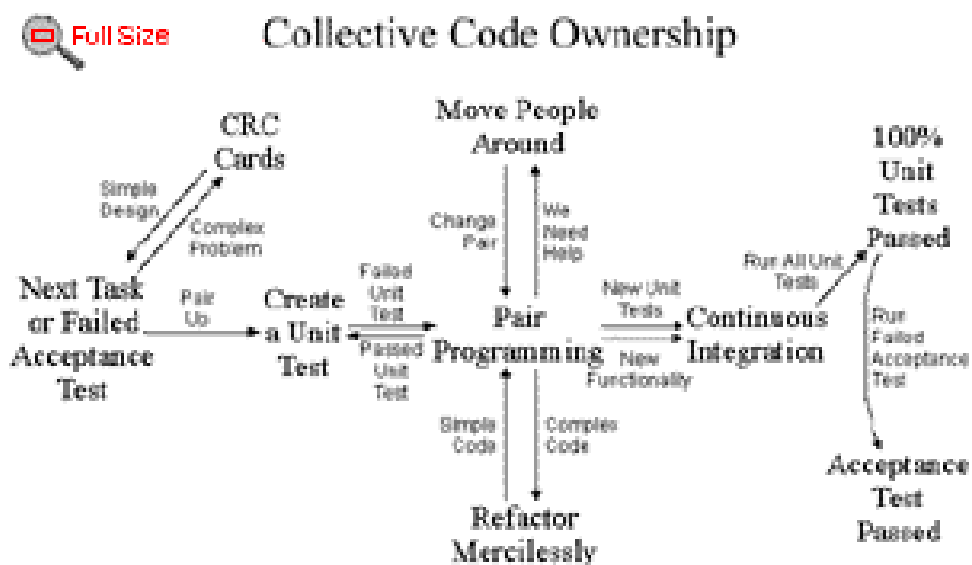


Figura.25 Propiedad colectiva del código

El diálogo que existe entre las personas involucradas en el proyecto permite determinar:

Ámbito: ¿Qué es lo que el software debe de resolver para que este genere valor?

Prioridad: ¿Qué debe ser hecho en primer lugar?

Composición de versiones: ¿Cuánto es necesario hacer para saber si el negocio va mejor con software que sin él? Una vez que el software aporte algo al negocio se debe tener lista las primeras versiones.

Fechas de versiones: ¿Cuáles son las fechas en la presencia del software o parte del mismo pudiese marcar la diferencia?

Estimaciones: ¿Cuánto tiempo lleva implementar una característica?

Consecuencias: Informar sobre las consecuencias de la toma de decisiones por parte del negocio. Por ejemplo el cambiar las bases de datos a Oracle.

Procesos: ¿Cómo se organiza el trabajo y el equipo?

Programación detallada: Dentro de una versión ¿Qué problemas se resolverán primero?

2.1.17.2. DISEÑO

El diseño es muy importante para un proyecto, se establecen mecanismos para mejorar continuamente a lo largo del proyecto, según se añadan funcionalidades al mismo.

Añade agilidad al proceso de desarrollo y evita que se mire demasiado hacia delante, desarrollando trabajos que aún no han estado programados.

El diseño debe ser simple, pero debe funcionar con todas las pruebas que se realicen, si es necesario borrar lo innecesario.

En esta etapa se realizan las metáforas para que el nombrado de las clases se mantengan, logrando la reutilización y comprensión de código, para tener el menor número de clases y objetos.

Una vez teniendo claro que es lo que se desea se escriben las tarjetas CRC para cada objeto, permitiendo que todo el equipo de desarrollo participe y aporte mejoras al sistema.

Si el diseño tiene que cambiar o eliminar código que no sirve, se realiza sin miedo es decir se refactoriza permitiendo que el sistema sea más sencillo.

2.1.17.3. DESARROLLO

El desarrollo es la parte más importante en el proceso de la programación extrema. Todos los trabajos tienen como objetivo que se programen lo más rápidamente posible, sin interrupciones y en dirección correcta.

Para lograr esto es importante que el cliente siempre esté disponible, que forme parte del equipo de desarrollo y este en todas las fases de XP. La idea es no perder el tiempo del cliente en crear una detalladísima especificación de requisitos, y evitar la entrega de un producto que no funcione.

En esta etapa se debe considerar que el código que se está creando debe mantener los estándares de codificación acordados, asegurando la consistencia y facilitando la comprensión y refactorización del código.

Para aumentar la calidad del proyecto la programación se realizará en parejas, y en cada momento existirá sólo una pareja de programadores integrando código. La integración debe ser frecuente, permitiendo que todos trabajen con la última versión y no existan diferencias

en el desarrollo, además se detectarán rápido las incompatibilidades, ya que no se integra el código al final del desarrollo.

Al usar la propiedad colectiva, permite que cualquier programador conozca cualquier parte de código del sistema, permitiendo que se fomente la contribución de ideas.

Además las pruebas unitarias se crean ya que deben ser codificadas antes que el código en sí, logrando tener más claro el objetivo a cumplir y que el código se desarrolle más rápido.

2.1.17.4. PRUEBAS

En XP hay una máxima que dice "Todo el código que puede fallar tiene que tener una prueba".

El código desarrollado debe tener pruebas unitarias, éstas deben ser sencillas, una vez creadas debe pasarlas para tener el release.

Es importante desarrollar pruebas por si se encuentran errores de codificación o bugs, para no volver a caer en los mismos.

Finalmente las pruebas de aceptación deben ser frecuentes y publicar los resultados obtenidos. Estas pruebas se las obtiene a partir de las historias de usuario escogidas para cada iteración, son conocidas como "pruebas de caja negra", y es el cliente quien verifica si se encuentran funcionando correctamente. Si la historia de usuario pasa la prueba de aceptación se la considera como completada.

A continuación se muestra una imagen del proceso de la metodología XP en cada una de las fases.

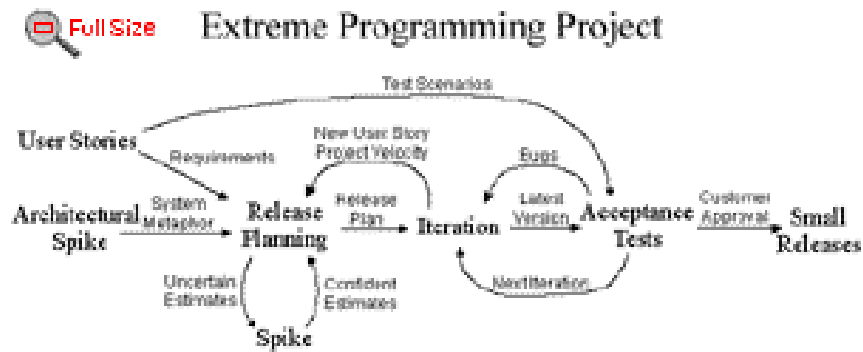


Figura.26 Fases de XP

2.1.18. VENTAJAS DE UTILIZAR LA METODOLOGÍA XP

- El cliente y el equipo de desarrollo realizan un trabajo conjunto.
- Los costos se minimizan significativamente, si existen cambios, ya que se desarrolla solo lo que realmente requiere el cliente.
- El trabajo innecesario se elimina, al utilizar la simplicidad en el desarrollo del sistema.
- Existe un diseño simple y fácil de entender, permitiendo que el desarrollo de código sea más rápido.
- Los requerimientos del cliente son bien entendidos por los desarrolladores, al estar el cliente presente en todo el proceso de desarrollo.
- Al escribir una prueba unitaria, se piensa en la forma correcta de utilizar un módulo que aún no existe.
- Cada componente del producto final se prueba y satisface los requerimientos del cliente.

- Cuando las pruebas unitarias fallan, o se descubre un defecto, los desarrolladores pueden revertir el código base a un estado libre de defectos, sin perder tiempo depurando.
- Al sentirse en un ambiente agradable de trabajo los desarrolladores se encuentran motivados y relajados al trabajar el tiempo necesario, que aportan muchas mejoras al proyecto.
- Disminución del riesgo de gestión: ya que el conocimiento del sistema se comparte entre varios programadores, hay menos riesgo para gestionar si un programador abandona el equipo.

Además la metodología XP supone:

- Las personas son claves en los procesos de desarrollo.
- Los programadores son profesionales y no necesitan supervisión.
- Los procesos se aceptan y se acuerdan, no se imponen.
- Desarrolladores y gerentes comparten el liderazgo del proyecto.
- El trabajo de los desarrolladores con las personas que conocen el negocio es regular, no puntual.

A todo esto, es importante recordar que:

“ninguna metodología hace el trabajo por sí sola, pero puede ayudar”.

2.1.19. LIMITACIONES

Muchos que están en desacuerdo con XP han incluido muchas críticas, sin embargo los que se encuentran a favor de ésta metodología dicen que son mal entendidos del desarrollo ágil.

- Si no existe confianza entre las parejas de programación, no se avanzará en el desarrollo del sistema. Muchos creen que lo que han desarrollado les pertenece y no lo comparten con los otros miembros del equipo y generalmente no se esfuerzan para que su pareja los entienda.
- Una de las críticas que tiene XP, es que no todos los programadores pueden hacer diseños fáciles, sencillos y extensibles.
- XP se volvería complicado si se desea trabajar con grupos mayores a 10 personas, ya que todas deben estar todas juntas es decir en el mismo centro de trabajo.
- Puede aumentar el riesgo de cambios del alcance, debido a la falta de documentación de requisitos detallada.

3. CAPITULO III

3.1. ARQUITECTURA MVC

3.1.1. DEFINICIÓN

Es un patrón de diseño de arquitectura de software principalmente en aplicaciones que manejan gran cantidad de datos y transacciones complejas donde se requiere una mejor separación de conceptos para que el desarrollo esté estructurado de una mejor manera, decrementando la duplicación de código, facilitando la programación en diferentes capas de manera paralela e independiente, y permitiendo que la aplicación sea más extensible. MVC es el patrón de diseño arquitectural para la capa de presentación.

3.1.2. CARACTERÍSTICAS DE MVC²⁹

Es así que tenemos: Modelo, Vista y Controlador.

- Los datos (que responden a unos modelos)
- La interfaz gráfica (la vista, como se van a representar esos datos)
- La lógica (el controlador, lo que se hace y como se hace)

²⁹<http://en.juantxu.net/doku.php/jee/3?do=index>

3.1.2.1. EL MODELO³⁰

Es la representación de la información que maneja la aplicación. Son los datos puros que puestos en contexto del sistema proveen de información al usuario o a la aplicación misma. Encapsula las reglas del negocio.

3.1.2.2. LA VISTA

Es la representación del modelo en forma gráfica disponible para la interacción con el usuario. Es una página HTML con contenido dinámico sobre el cual el usuario puede realizar operaciones. Es la interfaz de usuario.

3.1.2.3. EL CONTROLADOR

Es la capa encargada de manejar y responder las solicitudes del usuario, procesando la información necesaria y modificando el Modelo en caso de ser necesario. Gestiona el workflow de la aplicación. La arquitectura MVC separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones.

Si una misma aplicación debe ejecutarse tanto en un navegador estándar como un navegador de un dispositivo móvil, solamente es necesario crear una vista nueva para cada dispositivo; manteniendo el controlador y el modelo original. El controlador se encarga de aislar al modelo y a la vista de los detalles del protocolo utilizado para las peticiones (HTTP, consola de comandos, email, etc.). El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes de, por ejemplo, el tipo de gestor de bases de datos utilizado por la aplicación.

³⁰http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/rivera_l_a/capitulo2.pdf

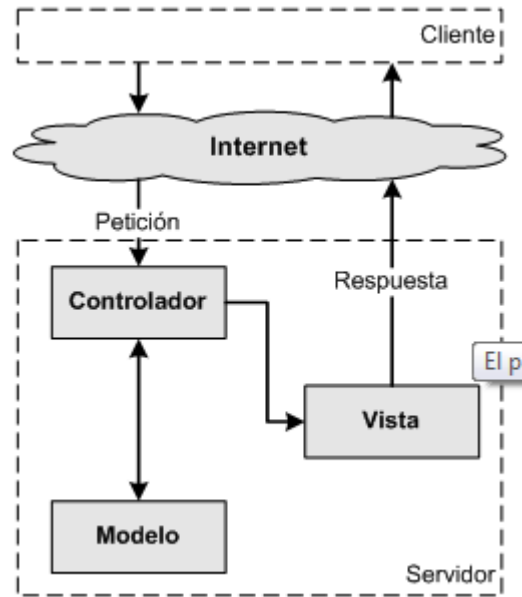


Figura.27 **Patrón MVC** : http://feeds.feedburner.com/librosweb_es

3.1.3. CICLO DE VIDA MVC

El ciclo de vida MVC empieza cuando el usuario hace una solicitud al controlador con información de algo que desea realizar. El controlador recibe una notificación de lo solicitado, gestiona el evento y delega la tarea a quien corresponda, es ahí cuando el Modelo empieza a trabajar, realiza las operaciones para manejar la información solicitada por el controlador, termina su labor y regresa la información resultante al controlador, éste redirige la información a la Vista la cual transforma estos datos en información entendible por el usuario.

El ciclo empieza nuevamente cada vez que el usuario quiere.

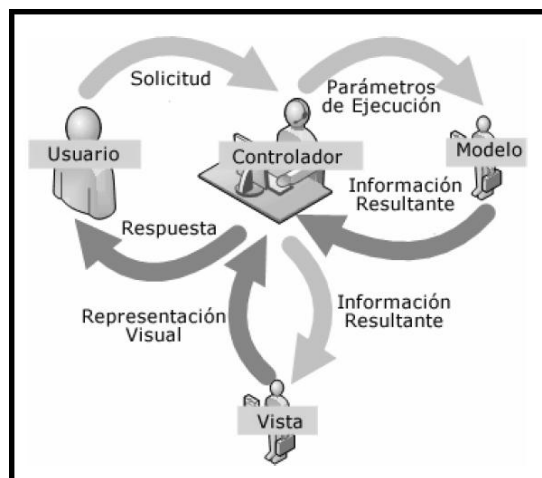


Figura.28 Ciclo de Vida MVC

3.1.4. FRAMEWORKS MVC

A continuación tenemos un listado de lo Frameworks MVC más comunes:

Lenguaje	Licencia	Nombre
Ruby	MIT	Ruby onRails
Ruby	MIT	Merb
Ruby	MIT	Ramaze
Java / J2ee	Apache	Grails
Java / J2ee	GPL	Interface Java Objects
Java / J2ee	LGPL	Framework Dinámica
Java / J2ee	Apache	Struts
Java / J2ee	Apache	Beehive
Java / J2ee	Apache	Spring
Java / J2ee	Apache	Tapestry
Java / J2ee	Apache	Aurora
Java / J2ee	Apache	JavaServerFaces
JavaScript	GPLv3	ExtJS 4
Perl	GPL	Mojolicious
Perl	GPL	Catalyst
Perl	GPL	CGI::Application
Perl	GPL	Gantry Framework

Perl	GPL	Jifty
Perl	GPL	Maypole
Perl	GPL	OpenInteract2
Perl	Comercial	PageKit
Perl	GPL	Cyclone 3
Perl	GPL	CGI::Builder
PHP	GPL	Self Framework (php5, MVC, ORM, Templates, I18N, Multiples DB)
PHP	LGPL	ZanPHP
PHP	LGPL	Tlalokes
PHP	GPL	SiaMVC
PHP	LGPL	Agavi
PHP	BSD	Zend Framework
PHP	MIT	CakePHP
PHP	GNU/GPL	KumbiaPHP
PHP	MIT	Symfony
PHP	MIT	QCodo
PHP	GNU/GPL	CodeIgniter
PHP	Otra	Kohana
PHP	MPL 1.1	PHP4ECore
PHP	BSD	PRADO
PHP	GNU	FlavorPHP
PHP	Apache 2.0	Yupp PHP Framework
PHP	BSD	Yii PHP Framework
PHP	GPL	Logick PHP Framework
PHP	GPL	Osezno PHP Framework
Python	ZPL	Zope3
Python	Varias	Turbogears
Python	GPL	Web2py
Python	BSD	Pylons
Python	BSD	Django
.NET	Castle Project	MonoRail
.NET	Apache	Spring .NET
.NET	Apache	Maverick .NET

.NET	<u>MS-PL</u>	ASP.NET MVC
.NET	Microsoft Patterns&Practices	User Interface Process (UIP) Application Block
AS3	Adobe Open Source	Cairngorm
AS3 y Flex	MIT License	CycleFramework

Tabla. 4 Frameworks MVC³¹

3.1.5. CONSIDERACIONES.

Se debe considerar lo siguiente a la hora de la utilización de una arquitectura MVC.

Es importante tomar en cuenta que tan grande y la complejidad que tiene la aplicación, para la utilización de la arquitectura MVC, ya que así permitirá que sea escalable.

Al desarrollar una aplicación con la arquitectura MVC, su funcionamiento es independiente facilitando el mantenimiento en caso de errores y las pruebas se hacen más sencillas.

Da la posibilidad de crear equipos altamente especializados.

Al tener dividida la aplicación en varias partes, los archivos a mantener y desarrollar se incrementan considerablemente.

3.2. TECNOLOGÍAS A USAR

3.2.1. BASE DE DATOS ORACLE.³²

Oracle es un sistema de gestión de base de datos objeto-relacional (o ORDBMS por el acrónimo en inglés de Object-Relational Data Base Management System), desarrollado por Oracle Corporation.

³¹<http://es.wikipedia.org/w/index.php?>

³²<http://www.orasite.com/index.php?>

El modelo relacional (de un modo sencillo) consiste en utilizar **tablas bidimensionales** para almacenar la información.

Consta de tres elementos básicos:

- Tablas
- Conjunto de operadores para manipular esas tablas
- Reglas de integridad

La estructura básica del modelo relacional tiene los siguientes elementos:

- **Relación:** En el modelo relacional se representa mediante una tabla con m filas y n columnas. Como las tablas son esencialmente relaciones, se utilizarán los términos matemáticos relación y tupla, en lugar de los términos tabla y fila.
- **Atributos:** Son las columnas de la tabla. Corresponden a las propiedades de las entidades. Cada uno de estos atributos puede tomar valores dentro de un rango determinado, que se llama dominio. Varios atributos pueden compartir un único dominio.
- **Dominio:** Rango de valores aceptable para un atributo dado. Este rango depende exclusivamente del atributo y va a condicionar los valores posibles dentro de cada celda de la tabla.
- **Tuplas:** Es el nombre que recibe cada una de las filas de la tabla.
- **Cardinalidad de la relación:** es el número m de tuplas de la relación.
- **Grado de la relación:** Es el número n de atributos que intervienen en la relación.

RDBMS posee una serie de subsistemas que se encargan de gestionar cada servicio.

Algunos de estos subsistemas son ⁱ:

- **Sistema de gestión de la memoria.** Encargado de decidir que parte de la memoria se dedica a cada tarea del RDBMS. Su función es que haya suficiente memoria para que el RDBMS funcione eficazmente y a la vez nunca dejar menos memoria de la que necesita el Sistema Operativo para que la máquina funcione.
- **Gestión de Entrada y Salida.** Para conseguir que los accesos a los datos sean adecuados.
- **Procesador de lenguajes.** Para interpretar las instrucciones SQL (o de otros lenguajes válidos) que los usuarios lanzan a la base de datos.
- **Control de procesos.** Gestiona los programas en ejecución necesarios para el funcionamiento de la base de datos.
- **Control de la red.** Para gestionar las conexiones a la base de datos desde la red y evitar problemas a la base de datos en caso de desconexión.
- **Control de transacciones**³³. Permite gestionar las transacciones (series de operaciones que se pueden anular o llevar a cabo al final).

Oracle es uno de los sistemas de bases de datos más completos, ya que permite realizar lo siguiente:³⁴

- Soporte de transacciones,
- Estabilidad,
- Escalabilidad y
- Soporte multiplataforma.

³³<http://creativecommons.org/licenses/by-nc-sa/2.0/legalcode>

³⁴<http://www.oracle.com/index.html>

- Garantiza la seguridad.
- Maximizar la disponibilidad.
- Visibilidad transparente.

3.2.1.1. ELEMENTOS DEL SERVIDOR ORACLE

El servidor Oracle está formado por dos elementos:

La instancia de la base de datos. Consta de **datos** (llamados estructuras de memoria) y de **procesos** en memoria (procesos background) necesarios para dar servicio a los usuarios de la base de datos. Puede haber más de una instancia si se distribuye la base de datos en más de una máquina. Cada instancia abre **una y sólo una** base de datos.

Ficheros en disco. Representan la base de datos en sí. Consta de:

Estructuras lógicas: Tablespaces, objetos del esquema de usuario.

Estructuras físicas: Los ficheros de datos almacenados en disco. Los ficheros de datos (asociados a los *tablespaces*), los ficheros *redo log* y los ficheros de control.

3.2.1.2. INSTANCIA ORACLE

Una instancia de Oracle comprende estructuras de memoria conocidas como **SGA (System Global Area)** y los procesos background de Oracle. La instancia de base de datos Oracle comprende también los “**datafiles**”, “**redo log files**” y los “**control files**”.

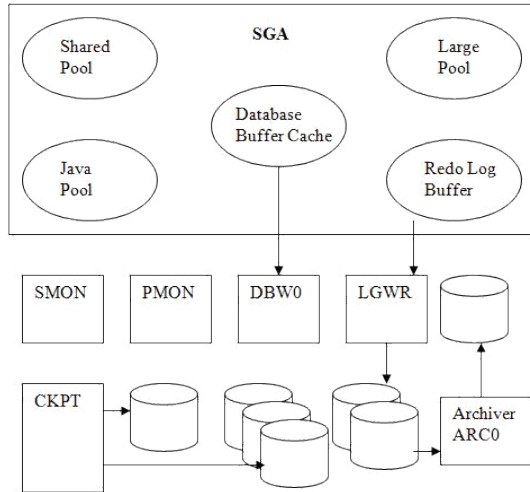


Figura.29 Instancia de Oracle

La comunicación con el servidor de base de datos se realiza usando SQL, entonces se obtiene la información procesando las sentencias SQL que el usuario ingresa.

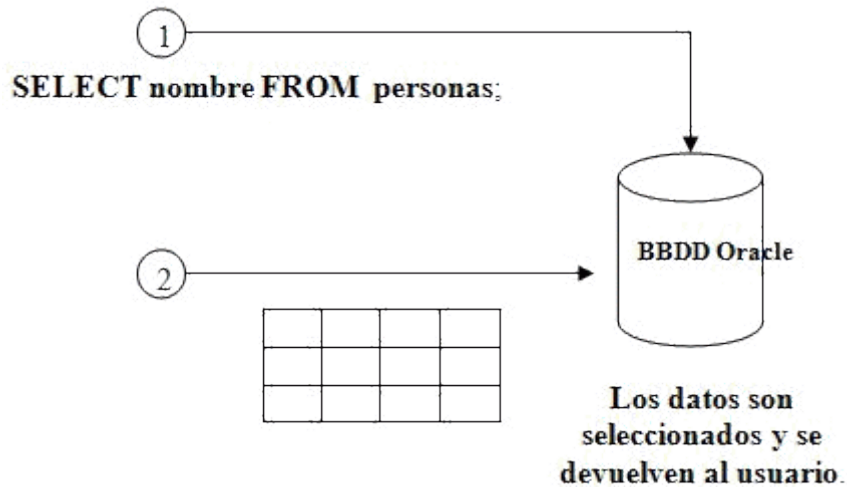


Figura.30 Acceso a la Base de Datos

3.2.1.3. HERRAMIENTAS ORACLE

A continuación tenemos algunas herramientas usadas para administrar una base de datos oracle.

- Oracle Universal installer (Instalador de Oracle)
- DatabaseConfigurationAssistant (Asistente para la configuración de oracle)

- DatabaseUpgradeAssistant (Actualizar base de datos)
- Oracle Net Manager (Para configurar la red de oracle)
- Oracle Enterprise Manager (Para administrar la base de datos)
- SQL* Plus admiSQL*PLUS (Para realizar sentencias SQL contra la base de datos)
- Recovery Manager (Para recuperar bases de datos)
- Data Pump (Para realizar copias de seguridad, antiguo imp/exp)
- SQL*Loader (Para realizar la carga de datos)

3.2.1.4. VERSIONES

Oracle tiene 6 ediciones a partir de la versión 10g Release 2:

- Oracle Database Enterprise Edition (EE).
- Oracle Database Standard Edition (SE).
- Oracle Database Standard Edition One (SE1).
- Oracle Database Express Edition (XE).
- Oracle Database Personal Edition (PE).
- Oracle Database Lite Edition (LE).

La única edición gratuita es la Express Edition, que es compatible con las demás ediciones de Oracle Database 10gR2 y Oracle Database 11g.

La base de datos Oracle soporta todos los tipos de datos relacionales estándares, así como también datos nativos como XML, texto, imágenes, documentos, audio, y datos espaciales.

El acceso a la información es realizado a través de interfaces estándares como SQL, JDBC, SQLJ, ODBC.Net, OLE.Net y ODP.Net, SQL/XML, XQuery y WebDAV. Los procedimientos almacenados pueden ser escritos en Java, PL/SQL.

3.2.2. JDeveloper³⁵

JDeveloper es un entorno de desarrollo integrado para programar aplicaciones o Applets en Java, es desarrollado por Oracle Corporation para los lenguajes Java, HTML, XML, SQL, PL/SQL, Javascript, PHP, Oracle ADF, UML y otros.

JDeveloper es de la Oracle y la diferencia con otros IDEs para programar Java, es que tiene integrado comunicación con drivers nativos para Base de Datos Oracle.

<http://es.debugmodeon.com/debate/cuales-son-las-ventajas-de-jdeveloper>

JDeveloper va más allá de ser un IDE para crear aplicaciones Java y lo que promueve es el concepto de IDE empresarial.

Para el desarrollador el ambiente, los menús, etc, son los mismos; no se requiere de utilizar otras herramientas, el IDE se ajusta de acuerdo al tipo de tecnologías a utilizar.

3.2.2.1. VERSIONES DE JDEVELOPER

Es un software propietario pero gratuito desde 2005.

Las primeras versiones de 1998 estaban basadas en el entorno JBuilder de Borland, pero desde la versión 9i de 2001 está basado en Java, no estando ya relacionado con el código anterior de JBuilder.

Además tenemos otras versiones estables como:

- Para JDK 6: 11.1.1.2.0 (noviembre de 2009)
- Para JDK 5: 10.1.3.5 (agosto de 2009).

³⁵<http://es.debugmodeon.com/debate/cuales-son-las-ventajas-de-jdeveloper>

JDeveloper cuenta con un servidor JavaEE embebido, el facilita toda la parte de pruebas y despliegues. Existe una versión de producción la cual viene con OC4J. Pero como tal la herramienta permite hacer despliegues a otros contenedores Java EE.

Además JDeveloper tiene ADF o ApplicationDevelopment Framework, un framework end-to-end para construir aplicaciones Java EE. Utilizando este framework en conjunto con JDeveloper se pueden crear aplicaciones basadas en MVC de manera productiva y basada en los estándares de la industria tales como EJB3, JPA, JSF.

3.2.2.2. CICLO DE VIDA DE JDEVELOPER

Jdeveloper cubre el ciclo de vida completo del desarrollo del diseño con la codificación, eliminar errores, pruebas, la optimización y perfilar a desplegar, todo en estilo corporativo.

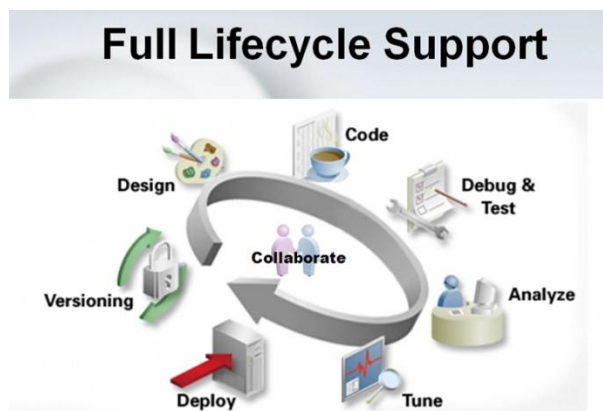


Figura.31 Ciclo de vida de JDeveloper

Esta plataforma disminuye mucho la elaboración de código, brinda ayuda por medio de asistentes que facilitan la elaboración de mucha programación, sin embargo no posee ayudas en cuanto a los servicios de relaciones con tablas de las bases de datos para mostrar los datos de manera esquematizada y para hacer comparaciones con los datos almacenados.

Oracle ha centrado gran parte de su estrategia a nivel de herramientas de desarrollo entorno a JDeveloper, esto quiere decir que el IDE cuenta con características que cubren todo el ciclo de vida de una aplicación (análisis basado en UML, desarrollo, prueba/debug, elementos para auditar y versionamiento), adicionalmente se puede hacer desarrollo orientado a base de datos (modelamiento), desarrollo para XML, desarrollo para SOA (BPEL, ESB, Webservices), desarrollo web (javaScript, HTML, JSF).

Experiencia de desarrollo con JDeveloper.

Se enlistan algunas experiencias al desarrollar con JDeveloper.

- Desarrollo productivo con JavaEE.
- Entorno Declarativo y Visual.
- Desarrollo integrado.
- Escoger una propuesta de desarrollo.

3.2.3. OC4J

The Oracle Application Server Containersfor J2EE (OC4J) standalone provee una distribución completa de entorno del servidor J2EE, se encuentra distribuido como un simple archivo .zip. Esta distribución incluye un servidor HTTPs, los servicios requeridos de J2EE, y las capacidades de Web Services, todos se ejecutan desde un proceso de Java.

3.2.3.1. Pre-requisitos

Para utilizar OC4Jstandalone, debe tener instalado en su sistema una Java2 Standard Edition (J2SE) versión 1.4 o posterior

3.2.4. Modelo Vista Controlador en JSF³⁶

El patrón MVC (Modelo Vista Controlador), permite separar la lógica de control, la lógica de negocio y la lógica de presentación.

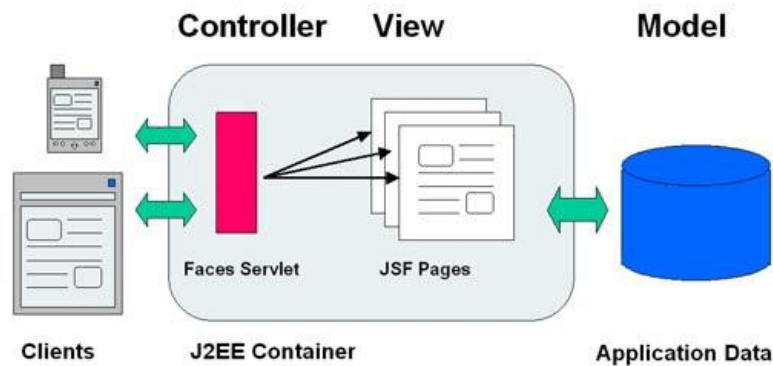


Figura.32 MVC en JSF

3.2.4.1. MODELO³⁷

Todas las aplicaciones *software* dejan a los usuarios manipular ciertos datos que proceden de una realidad sobre la que se pretende actuar, como supermercados, itinerarios de viaje, o cualquier dato requerido en un dominio problemático particular. A estos datos en estado puro, que representan el estado de la realidad se les llama modelo: modelan la parte de la realidad sobre la que se desea actuar.

El modelo, pues, es el objeto que representa y trabaja directamente con los datos del programa: gestiona los datos y controla todas sus transformaciones. El modelo no tiene conocimiento específico de los diferentes controladores y/o vistas, ni siquiera contiene

³⁶http://www.programacion.com/articulo/integracion_de_jsf_spring_e_hibernate_para_crear_un_a_aplicacion_web_del_mundo_real_307/3

³⁷Jsf.pdf

referencias a ellos. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el modelo y sus vistas, y notificar a las vistas cuándo deben reflejar un cambio en el modelo.

3.2.4.2. VISTA

La vista es el objeto que maneja la presentación visual de los datos gestionados por el Modelo. Genera una representación visual del modelo y muestra los datos al usuario.

Interacciona con el modelo a través de una referencia al propio modelo.

En el ejemplo básico, la vista está manipulada a través de las páginas JSF.

3.2.4.3. CONTROLADOR

El controlador es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el modelo. Entra en acción cuando se realiza alguna operación, ya sea un cambio en la información del modelo o una interacción sobre la

Vista. Se comunica con el modelo y la vista a través de una referencia al propio modelo.

Además, JSF opera como un gestor que reacciona ante los eventos provocados por el usuario, procesa sus acciones y los valores de estos eventos, y ejecuta código para actualizar el modelo o la vista.

3.3. ARQUITECTURA DEL SISTEMA DE GESTIÓN DE AHORRO PARA LA ASOCIACIÓN DE PROFESORES DE FICA.

El Sistema de Gestión de Ahorro para la Asociación de Profesores de la FICA, se encuentra desarrollado en la tecnología JSF JavaServer Faces que constituye un marco de trabajo (*framework*) de interfaces de usuario del lado de servidor para aplicaciones web basadas en tecnología Java y en el patrón MVC (Modelo Vista Controlador).

El lenguaje utilizado es Jdeveloper el cual trabaja muy bien con JSF, se utiliza páginas JSP para las funciones de la Vista, Faces Servlet para cumplir las funciones del Controlador y para la parte del Modelo se hace uso de EJB's (Enterprise Java Beans)

La Base de Datos utilizada es Oracle 10g, la cual se interconecta muy bien con Jdeveloper.

3.3.1. CICLO DE VIDA MVC PARA SIGFAP

El usuario hace una solicitud al sistema ingresando los datos de autenticación esta solicitud es atendida por el controlador.

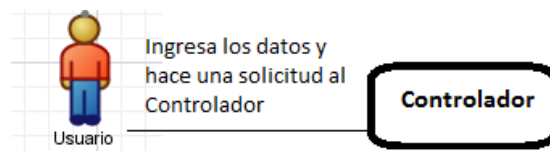


Figura.33 Solicitud de usuario en la Vista- MVC

El controlador gestiona este evento y se comunica con el Modelo.

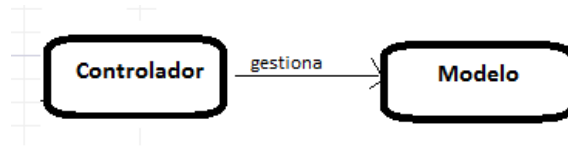


Figura.34 Gestión del Controlador

El Modelo empieza a trabajar y reacciona devolviendo los datos que fueron solicitados.

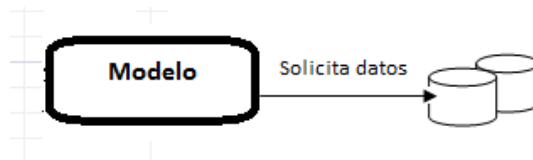


Figura.35 Gestión del Modelo

Estos datos nuevamente son manejados por el Modelo y son redirigidos hacia el Controlador.

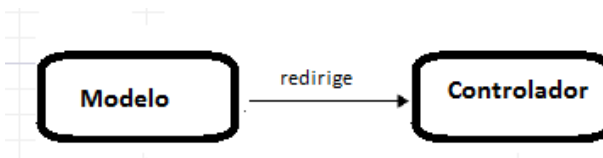


Figura.36 Información redirigida del Modelo al Controlador.

Los datos son transformados de tal forma que el usuario los entienda.

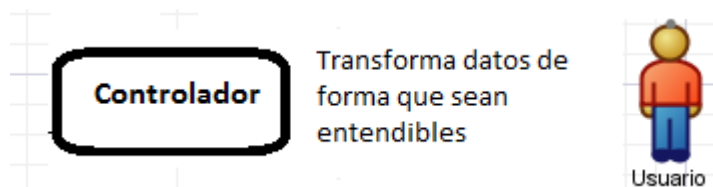


Figura.37 Datos transformados en la Vista

4. CAPITULO IV

4.1. APLICACIÓN DE LA METODOLOGÍA XP AL PROYECTO

En el proyecto se va a tomar como ejemplo una historia de usuario, la cual se tomará para demostrar los demás artefactos que tiene la metodología XP.

4.1.1. PLANIFICACIÓN

Sistema de Gestión de Fondos para la Asociación de Profesores de la FICA.

Se desea que el sistema gestione los fondos que los profesores de la FICA tienen, sus aportes, préstamos, pagos mensuales, entre otros. Para esto al reunirse con el usuario se han definido las siguientes historias de Usuario.

4.1.1.1. HISTORIAS DE USUARIO.

A continuación se muestra un listado de las historias de usuario del proyecto “Gestión de fondos para la Asociación de Profesores de la FICA”.

1. Registro de Asientos Contables.
2. Otorgar créditos a los socios de la Asociación de profesores.
3. Gestión de Plan de cuentas.
4. Ingreso de nuevos usuarios.
5. Gestión de un nuevo Periodo Contable.

6. Ingresar tipo de cuentas
7. Ingreso de aportes de los socios
8. Registro de pagos de los socios.
9. Generar archivos de aportes.
10. Obtener reportes
11. Gestión de datos de los usuarios
12. Ingreso de interés
13. Control de acceso de usuarios
14. Revisar estado de cuenta.
15. Edición de mis datos.
16. Visualización de reportes.
17. Ganancias generadas.
18. Mayorización.
19. Revisar estado de cuenta de cada socio.
20. Cierre de periodo contable.
21. Pago total de préstamo.
22. Pago de Interés.

4.1.1.1.1. Descripción de las historias de usuario

La Historia de usuario correspondiente a: Registro de Asientos Contables.

Historia de Usuario	
Número: 1	Usuario: Administrador
Nombre historia: Registro de asientos contables	
Prioridad en negocio: Alta	Riesgo en desarrollo: Alta
Puntos estimados:	Iteración asignada: 1
Programador responsable: Lorena Guzmán	
Descripción: Como administrador quiero registrar los asientos contables en el libro diario para poder tener un control de los ingresos y egresos de la asociación.	
Observaciones:	

Figura.38 Historia de Usuario Registro de Asientos Contables

4.1.1.2. ACTIVIDADES

Tarea	
Número tarea: 1	Número historia: 1
Nombre tarea: Diseño de interfaz para ingreso de asientos contables	
Tipo de tarea : Desarrollo	Puntos estimados: 1
Fecha inicio: 2010	Fecha fin: 2010
Programador responsable: Equipo XP	
Descripción: Se diseña una interfaz la cual tenga un encabezado y un detalle del asiento contable. En el encabezado se debe determinar un identificador, fecha, un concepto por el cual se registra, total de debe y haber. El detalle contendrá la cuenta contable, debe y haber. Debe permitir ingresar varios detalles para el asiento contable que se deben mostrar en una lista, para lo cual se debe habilitar un botón. Además debe tener un botón que permita guardar los datos.	

Figura.39 Actividad 1 para la Historia de Usuario 1

Tarea	
Número tarea: 2	Número historia: 1
Nombre tarea: Registro de Asientos contables	
Tipo de tarea : Desarrollo	Puntos estimados: 1.5
Fecha inicio: 2010	Fecha fin: 2010
Programador responsable: Equipo XP	
<p>Descripción:</p> <p>Registra los datos correspondientes a un asiento contable como son: fecha, concepto, y los detalles correspondientes: cuenta contable, valor del debe y valor del haber.</p>	

Figura.40 Actividad 2 para la Historia de Usuario 1

Tarea	
Número tarea: 3	Número historia: 1
Nombre tarea: Actualizar totales	
Tipo de tarea : Desarrollo	Puntos estimados: 0.5
Fecha inicio:	Fecha fin:
Programador responsable:	
<p>Descripción:</p> <p>Cada vez que ingrese un detalle en el asiento se debe ir actualizando el valor del debe o haber dependiendo de en cual ingreso la cantidad.</p>	

Figura.41 Actividad 3 para la Historia de Usuario 1

Tarea	
Número tarea: 4	Número historia: 1
Nombre tarea: Adaptar la Base de Datos para que permita el almacenamiento de los asientos contables en un libro diario.	
Tipo de tarea : Desarrollo	Puntos estimados: 1
Fecha inicio:	Fecha fin:
Programador responsable: Paco	
<p>Descripción:</p> <p>Se debe crear las relaciones entre la cabecera de un asiento contable con sus detalles, además debe existir la relación de los detalles del asiento contable con el plan de cuentas.</p>	

Figura.42 Actividad 4 para la Historia de Usuario 1

Tarea	
Número tarea: 5	Número historia: 1
Nombre tarea: Guardar datos en Base de Datos	
Tipo de tarea : Desarrollo	Puntos estimados: 1
Fecha inicio:	Fecha fin:
Programador responsable: Paco	
<p>Descripción:</p> <p>Una vez ingresados los datos tanto de la cabecera como del detalle del asiento contable, se guardan los datos en la BDD tanto del libro como del detalle, y se actualizan los valores en los balances de acuerdo a la cuenta que ha sido afectada.</p>	

Figura.43 Actividad 5 para la Historia de Usuario 1

Tarea	
Número tarea:6	Número historia:1
Nombre tarea: Guardar datos en Base de Datos	
Tipo de tarea : Modificación	Puntos estimados:0,5
Fecha inicio:	Fecha fin:
Programador responsable:	
<p>Descripción:</p> <p>Al guardar los datos de un asiento contable en la BDD, además de actualizarse los balances, deben actualizarse el libro de mayorización, de acuerdo a la cuenta contable.</p>	

Figura.44 Actividad 6 para la Historia de Usuario 1

4.1.1.3. PLAN DE ENTREGAS.

De acuerdo a las historias de Usuario consideradas en el sistema se ha realizado el siguiente plan de entregas, el cual tiene las iteraciones para cada tarea. Este plan se encuentra desarrollado tomando en cuenta el esfuerzo y la prioridad de cada Historia de Usuario.

Nombre de Historia	Prioridad	Esfuerzo
Historia 1 Registro de Asientos Contables.	Alta	4
Historia 2 Otorgar créditos a los socios de la Asociación de profesores.	Alta	4
Historia 3 Ingresar Plan de cuentas.	Alta	3
Historia 4 Gestión de nuevos usuarios.	Media	2
Historia 5 Gestión de un nuevo Periodo Contable.	Media	1
Historia 6 Ingresar tipo de cuentas	Media	1
Historia 7 Ingreso de aportes de los socios	Alta	3

Historia 8 Registro de pagos de los socios.	Alta	4
Historia 9 Generar archivos de aportes.	Media	3
Historia 10 Obtener reportes	Baja	1
Historia 11 Gestión de datos	Baja	2
Historia 12 Ingreso de interés	Media	1
Historia 13 Control de acceso de usuarios	Alta	2
Historia 14 Revisar estado de cuenta.	Media	2
Historia 15 Gestión de mis datos.	Baja	2
Historia 16 Visualización de reportes.	Baja	1
Historia 17 Ganancias generadas.	Media	2

Tabla. 5 Prioridad de las Historias de usuario

Las siguientes historias de usuario se definieron casi a la mitad del proyecto, se crearon al ver la necesidad de ser implementadas.

Nombre de Historia	Prioridad	Esfuerzo
Historia 18 Mayorización.	Media	3
Historia 19 Revisar estado de cuenta de cada socio.	Baja	1
Historia 20 Cierre de periodo contable.	Media	4
Historia 21 Pago total de préstamo.	Media	3
Historia 22 Pago de Interés.	Media	4

Tabla. 6 Historias de usuario definidas en una fecha posterior.

Evaluando cada historia de usuario se puede determinar en qué iteración se desarrollará la misma, a partir de esto se determina las fechas de inicio y fin para las entregas. A continuación se muestra la tabla de entregas.

Iteraciones	Historia	Prioridad	Esfuerzo	Fecha Inicio	Fecha Final
Iteración 1	Historia 1	Alta	4	18-01-2010	12-02-2010
Iteración 2	Historia 2.	Alta	4	15-02-2010	12-03-2010
Iteración 3	Historia 3	Alta	3	15-03-2010	2-04-2010
Iteración 4	Historia 7	Alta	3	8-04-2010	28-04-2010
Iteración 5	Historia 8	Alta	4	29-04-2010	19-05-2010
Iteración 6	Historia 13	Alta	2	20-05-2010	2-06-2010
Iteración 7	Historia 4	Media	2	7-06-2010	2-07-2010
	Historia 5.	Media	1		
	Historia 6	Media	1		
Iteración 8	Historia 9	Media	3	5-07-2010	23-07-2010
Iteración 9	Historia 12	Media	1	26-07-2010	13-08-2010
	Historia 14	Media	2		
Iteración 10	Historia10	Baja	1	18-08-2010	08-09-2010
	Historia 11	Baja	2		
Iteración 11	Historia 15	Baja	2	9-09-2010	29-09-2010
	Historia 16	Baja	1		
Iteración 12	Historia 17	Media	2	30-09-2010	13-10-2010
Iteración 13	Historia 18.	Media	3	14-10-2010	3-11-2010
Iteración 14	Historia 20.	Media	4	15-11-2010	10-12-2010
Iteración 15	Historia 21	Media	3	13-11-2010	7-01-2010
Iteración 16	Historia 22	Media	4	10-01-2010	4-02-2010
Iteración 17	Historia 19.	Baja	1	7-02-2010	11-02-2010

Figura.45 Fechas de entregas para cada una de las historias de Usuario

4.1.1.4. TRABAJO EN PAREJAS

No se aplica para el proyecto, ya que es una investigación personal de trabajo de tesis.

4.1.1.5. PROPIEDAD COLECTIVA DEL CÓDIGO.

Al trabajar individualmente el código no se ha compartido durante todo el diseño y desarrollo del sistema, pero si existen sugerencias o cambios en el código cualquier persona relacionada con el software le puede modificar el código de tal forma que no es prescindible la presencia de una sola persona.

Para que cualquier persona trabaje en el código se utilizó: JSF utilizando EJB's

4.1.2. DISEÑO

4.1.2.1. METÁFORA

La Asociación de Profesores de la Facultad de Ingeniería en Ciencias Aplicadas mantiene los siguientes fondos económicos, que se alimentan con las aportaciones mensuales de sus socios: aportes normales, caja de ahorros y pagos de préstamos.

Estas aportaciones permiten que se realicen nuevos préstamos a los socios para lo cual se hace un cálculo de las cuotas mensuales y son presentados en una tabla de amortización la cual es generada de acuerdo a el monto, plazo e interés.

Registrar los asientos contables de las transacciones que se hagan mensualmente, ya sean compras, pagos de préstamos, aportaciones, entre otros. Para esto es necesario tener un plan de cuentas.

Se debe tener un reporte de todas las transacciones realizadas es decir los balances los cuales son presentados cada cierto periodo contable, este periodo plantea un lapso de tiempo sabiendo su fecha inicial y final.

Todas estas tareas deben ser realizadas por un administrador.

El socio únicamente tendrá acceso a los reportes de las cuentas y su estado de cuenta.

4.1.2.2. CRC. Tarjetas de Responsabilidad Colaboración.

Usaremos una tarjeta de Responsabilidad y Colaboración para un Asiento Contable que es la historia de usuario la cual está en estudio.

En esta historia de usuario se necesita realizar 2 CRC, ya que se refiere a un asiento contable de acuerdo al análisis realizado se trata de una tabla Maestro Detalle, su descripción a continuación:

4.1.2.2.1. Tarjeta CRC Libro Diario

NOMBRE DE CLASE:
LIBRO DIARIO
RESPONSABILIDADES:
<p>Campos:</p> <p style="padding-left: 40px;">ID_LIBRO</p> <p style="padding-left: 40px;">CONCEPTO</p> <p style="padding-left: 40px;">FECHA</p> <p style="padding-left: 40px;">DEBE</p> <p style="padding-left: 40px;">HABER</p> <p>Métodos:</p> <p style="padding-left: 40px;">Ingresar_Datos()</p> <p style="padding-left: 40px;">ingresarLibro(String dias, String mes, String anio, String idPeriodo, String concepto)</p>
COLABORACIÓN:
<p>DETALLE_LIBRO_DIARIO</p> <p style="padding-left: 40px;">setDetalle(DetalleLibro detalle)</p> <p style="padding-left: 40px;">eliminarDetalle(String idDetalle)</p> <p style="padding-left: 40px;">String guardarDetalle(List listadetalle)</p> <p style="padding-left: 40px;">ingresarDetalle(Long id, String cuenta, Double debe, Double haber)</p>

Figura.46 Tarjeta CRC para el Libro Diario

4.1.2.2.2. Tarjeta CRC Detalle Libro Diario

NOMBRE DE CLASE:
DETALLE LIBRO DIARIO
RESPONSABILIDADES:
<p>Campos:</p> <p style="padding-left: 40px;">ID_DETALLE_LIBRO</p> <p style="padding-left: 40px;">ID_CUENTA</p> <p style="padding-left: 40px;">DEBE</p> <p style="padding-left: 40px;">HABER</p> <p>Métodos:</p> <p style="padding-left: 40px;">StringguardarDetalle(Listlistadetalle)</p> <p style="padding-left: 40px;">ingresarDetalle(Long id,LongidDetalle,Stringcuenta,Doubledebe, Double haber)</p>
COLABORACIÓN:

Figura.47 Tarjeta CRC para el Detalle de Libro Diario

4.1.3. DESARROLLO

4.1.3.1. BASE DE DATOS

4.1.3.1.1. Diagrama E-R

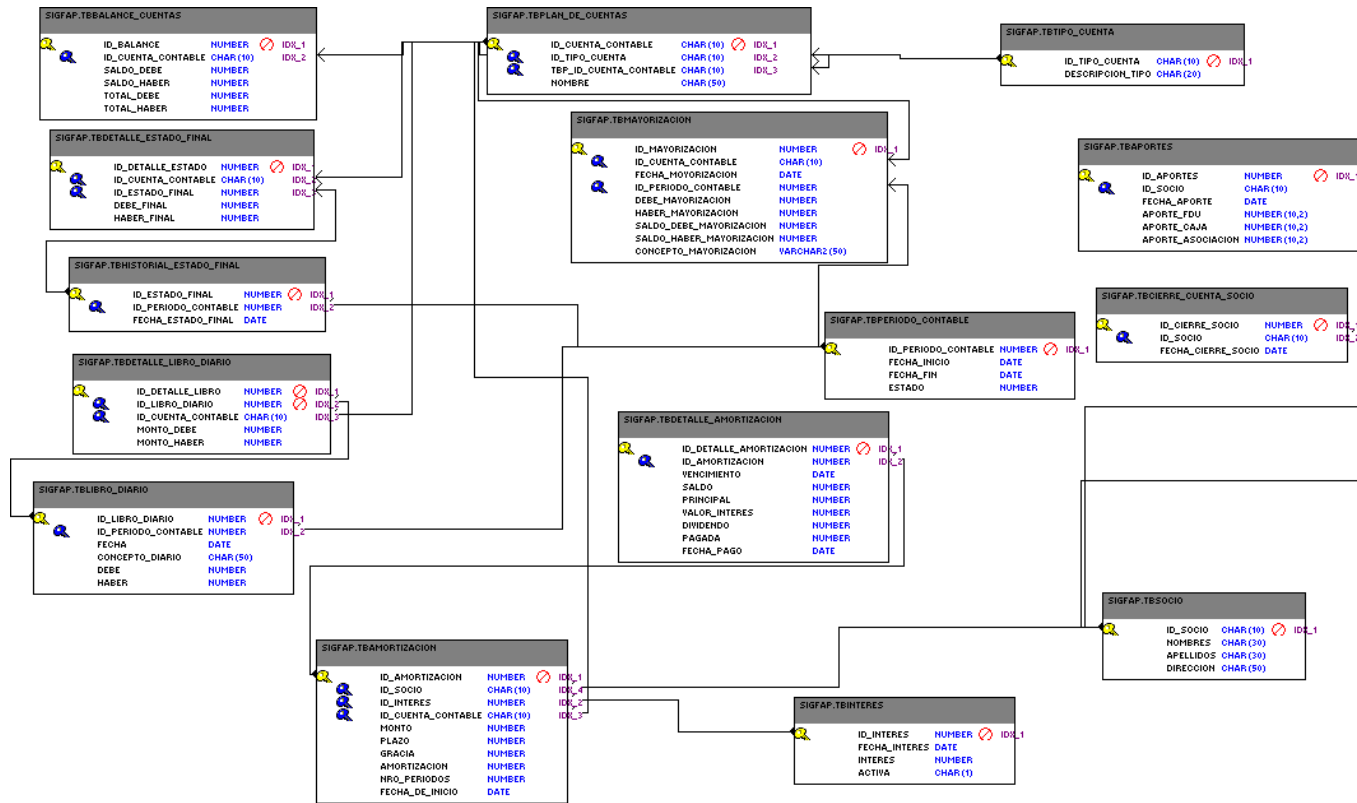


Figura.48 Diagrama E-R de SIGFAP

4.1.3.2. Prototipos

4.1.3.2.1. Ventana de Libro Diario en el Release 1.

INGRESE LOS DATOS DEL LIBRO DIARIO

Id Libro Diario	<input type="text"/>
Id Periodo Contable	<input type="text"/>
Concepto	<input type="text"/>
Fecha	<input type="text"/> <input type="text"/> <input type="text"/>
Haber	0
0	

INGRESO DEL DETALLE

Id Cuenta Contable	Debe	Haber
<input type="text"/>	<input type="text"/>	<input type="text"/>
	0	0

Figura.49 Ventana de Ingreso de datos de un Asiento contable en primer release

4.1.3.2.2. Ventana de Ingreso de Datos de Libro Diario en el último Release.

INGRESE LOS DATOS DEL LIBRO DIARIO

Id Periodo Contable	<input type="text" value="4"/>
Concepto	<input type="text"/>
Fecha	<input type="text"/> ...
Debe	Haber
0	0

INGRESO DEL DETALLE

Id Cuenta Contable	Debe	Haber
ACTIVO	<input type="text"/>	<input type="text"/>
	0	0

Id	Cuenta	Nombre	Debe	Haber
-----------	---------------	---------------	-------------	--------------

Figura.50 Ventana de Ingreso de datos de un Asiento contable en último release

4.1.4. PRUEBAS

4.1.4.1. PRUEBA DE ACEPTACIÓN.

Gestión de Fondos

Especificación de Prueba: Introducción de Asientos Contables

Historia 1

1. Descripción

Este documento cubre el conjunto de pruebas funcionales relacionadas con la historia de usuario: *Registro de Asientos Contables* [Historia 1]

En esta historia hay que comprobar la introducción de asientos contables preferente en la base de datos. Si la sintaxis del asiento contable no es correcta (no sigue el formato indicado o los valores son incorrectos con respecto del registro de comprobación) se informa al usuario y no se insertan los asientos incorrectos en la base de datos.

También debe comprobar, en el caso de que la introducción de asientos contables sea correcta, que el proceso de introducción sea correcto y los asientos sean almacenados en la base de datos.

Un asiento contable consta de una cabecera de asiento y varios detalles de asientos, y puede ocurrir que alguno de ellos sea válido mientras que otros no. En el proceso no se introduce ningún detalle de asiento contable y se notifica al usuario cuáles son incorrectos para que se lo comunique al cliente y éste verifique los datos incorrectos.

2. Introducción correcta de asientos contables

2.1. Descripción

El administrador una vez haya entrado en el sistema seleccionará la opción del menú “Asientos contables” – “Nuevo”. En esta ventana el administrador ingresará los datos correspondientes a un asiento contable si no hay ningún error de procesado en el asiento (sintaxis correcta y los datos son válidos) se avisará al administrador de la introducción satisfactoria y se guardarán los asientos contables en la base de datos.

2.2. Condiciones de ejecución

El administrador deberá estar dado de alta en el sistema.

Deben haberse ingresado las cuentas al plan de cuentas.

Debe haber ingresado los datos de un periodo contable.

2.3. Entrada

- El administrador introducirá su login y password.
- Del menú principal seleccionará “Asientos contables” – “Nuevo”.
- En esta ventana se mostrarán: un cuadro de texto en el cual introducirá el concepto.
- Debe seleccionar la fecha de ingreso del asiento contable.
- El valor del debe y haber se actualizarán dependiendo de los valores en los detalles ingresados.

2.4. Resultado esperado

Se encuentra listo los datos de la cabecera del asiento contable para ingresar los detalles del asiento.

2.5. Evaluación de la prueba

Prueba satisfactoria.

3. Introducción correcta de detalles de asientos contables

3.1. Descripción

El administrador una vez haya entrado en el sistema seleccionará la opción del menú “Asientos contables” – “Nuevo”. En esta ventana el administrador una vez ingresados los datos correspondientes a un asiento contable si no hay ningún error de procesado en el asiento (sintaxis correcta y los datos son válidos) están listos para poder ingresar los detalles de un asiento contable. Se avisará al administrador de la introducción satisfactoria y se guardarán los asientos contables en la base de datos.

3.2. Condiciones de ejecución

Debe haberse ingresado la cabecera de un asiento contable

3.3. Entrada

- El administrador introducirá su login y password.
- Del menú principal seleccionará “Asientos contables” – “Nuevo”.
- En esta ventana se mostrarán, el administrador ingresa los datos de un asiento contable.
- Para el ingreso de los detalles de un asiento contable, escogerá una cuenta contable del listado de cuentas.
- Debe ingresar el valor del debe o el valor del haber.
- Habrá un botón Ingresar, este permite registrar los datos del detalle.
- Cada vez que se ingrese un valor del detalle del asiento contable se irá actualizando los valores del debe y haber del asiento.

- Debe existir un botón para Guardar, internamente se debe analizar los valores del debe y haber y si son iguales continúa.
- Internamente se genera el identificador del asiento contable, y se guarda.
- Los datos se procesan internamente y se muestra un mensaje indicando que el asiento ha sido guardado correctamente.
- El proceso de introducción de asientos contables se considera como finalizado.

3.4. Resultado esperado

Tras la introducción de asientos contables y sus detalles, si el procesado ha sido correcto, en la base de datos aparecerán los datos de los nuevos asientos contables y sus detalles, y se actualizan los valores de mayorización y balances.

3.5. Evaluación de la prueba

Prueba satisfactoria.

4. Introducción de asientos contables con errores

4.1. Descripción

El administrador una vez haya entrado en el sistema seleccionará la opción del menú “Asientos contables” – “Nuevo”. En esta ventana el administrador ingresará los datos correspondientes a un asiento contable si existe un error de procesado en el asiento (sintaxis incorrecta o los datos no son válidos) se muestra un mensaje de error al administrador.

4.2. Condiciones de ejecución

El administrador deberá estar dado de alta en el sistema.

Deben haberse ingresado las cuentas al plan de cuentas.

4.3. Entrada

- El administrador introducirá su login y password.
- Del menú principal seleccionará “Asientos contables” – “Nuevo”.
- En esta ventana se mostrarán: un cuadro de texto en el cual introducirá el concepto.
- Debe seleccionar la fecha de ingreso del asiento contable.
- El valor del debe y haber se actualizarán dependiendo de los valores en los detalles ingresados.
- Si el administrador desea guardar los datos del asiento contable sin haber ingresado los detalles.
- Si el administrador guarda los datos y el valor del debe y haber no son iguales.

4.4. Resultado esperado

- Al ingresar valores erróneos o vacíos no permiten que ingrese los valores del detalle, se muestra un mensaje de error el cual debe ser corregido para permitir el ingreso de los detalles del asiento contable.
- Si al guardar los valores del debe y haber no son iguales, se muestra un mensaje de error, el cual pide rectificación en los valores de los detalles, para poder guardar el asiento.

4.5. Evaluación de la prueba

Prueba satisfactoria.

5. Introducción de detalles de asientos contables erroneos.

5.1. Descripción

El administrador una vez haya entrado en el sistema seleccionará la opción del menú “Asientos contables” – “Nuevo”. En esta ventana el administrador ingresará los datos correspondientes a un asiento contable si no hay ningún error de procesado en el asiento (sintaxis correcta y los datos son válidos), permite el ingreso de los detalles del asiento contable, si existe algún error en el ingreso de los detalles se debe mostrar un mensaje de error.

5.2. Condiciones de ejecución

Debe haberse ingresado el asiento contable.

5.3. Entrada

- El administrador introducirá su login y password.
- Del menú principal seleccionará “Asientos contables” – “Nuevo”.
- El administrador ingresa los datos de un asiento contable.
- Para el ingreso de los detalles de un asiento contable, escogerá una cuenta contable del listado de cuentas.
- Debe ingresar el valor del debe o el valor del haber.
- Habrá un botón Ingresar, este permite registrar los datos del detalle.
- Si el administrador ha ingresado valores tanto al debe como al haber, se detiene el proceso.

5.4. Resultado esperado

- Si ha ingresado valores en el debe y haber del detalle del asiento contable, debe mostrarse un mensaje que el ingreso ha sido erróneo y no se ingresa el detalle,

permitiéndole al administrador que ingrese nuevamente los datos de detalle del asiento contable

5.5. Evaluación de la prueba

Prueba satisfactoria.

5. CAPITULO V

5.1. CONCLUSIONES

- La metodología XP es una muy buena alternativa para el desarrollo de software, ya que da la posibilidad de ir definiendo más requerimientos a medida que va avanzando el proyecto, permitiéndole que sea escalable.
- Los desarrolladores al tener un buen ambiente de trabajo, desarrollan software de buena calidad en menor tiempo.
- Al trabajar con el cliente durante todo el desarrollo de software, la definición de los requerimientos son más entendibles y por ende mucho más fácil el desarrollo.
- Las historias de usuario son una muy buena estrategia para definir claramente que es lo que desea que realice el sistema.
- Las pruebas de aceptación permiten afianzar lo que especificó el cliente en las user histories, ya que se escriben como debe funcionar el sistema ante el ingreso de datos erróneos o bien cuando el ingreso de los datos son exitosos.
- Existen muy pocas fuentes de información acerca de los pasos a seguir para la aplicación de la metodología XP en el desarrollo de software.
- La descripción de los artefactos de la metodología XP es corta en la mayoría de las fuentes de información en Internet, únicamente hacen mención a éstos.
- La herramienta JDeveloper utilizada para el desarrollo del sistema SIGFAP, no fue la más adecuada para aplicar la metodología XP, que propone el desarrollo en

menor tiempo, ya que al no disponer de mucha documentación de su utilización, se presentaron errores que hicieron difícil el correcto avance del desarrollo del sistema.

5.2. RECOMENDACIONES

- Para el desarrollo de sistemas utilizando la metodología XP, se recomienda utilizar otro IDE de Java, puede utilizar Netbeans o Eclipse y si es necesario utilizar otra Base de Datos que sea mucho más compatible con estos lenguajes.
- Aplicar la metodología XP para el desarrollo de software, ya que los artefactos que utiliza para describir el funcionamiento del sistema son sencillos y fáciles de seguir.
- Se aplique el trabajo en Parejas en el desarrollo de software, ya que el conocimiento que uno de los dos tiene ayudará al otro para que vaya ganando experiencia, y el menos experimentado ayuda al otro encontrando soluciones óptimas y el proyecto avance más rápido.

5.3. BIBLIOGRAFÍA

Metodologías de Desarrollo de Software Ágiles.

- http://es.wikipedia.org/wiki/Desarrollo_de_software
- <http://www.cenitec.com.mx/Manifiesto.pdf>
- <http://seccperu.org/files/Metodologias%20Agiles.pdf>
- <http://www.willydev.net/descargas/prev/TodoAgil.Pdf>
- Metodologiasagiles.pdf
- agil.pdf.- Instituto Nacional de Tecnologías de la Comunicación
- schenone-tesisdegradoingenieriainformatica.pdf
- http://tratandodeentenderlo.blogspot.com/2010_06_01_archive.html
- http://en.wikipedia.org/wiki/Agile_software_development
- <http://www.ambysoft.com/unifiedprocess/agileUP.html>
- http://en.wikipedia.org/wiki/Agile_software_development
- <http://everac99.spaces.live.com/blog/cns!B2296C467C188917!659.entry.htm>
- <http://blog.tercerplaneta.com/2007/05/lean-product-develepment.html>
- Articulo TecnicoMetodologias Desarrollo.doc

Metodología de Programación Extrema

- Deigote's Blog
- <http://procesosdesoftware.wikispaces.com/METODOLOGIA+XP>
- <http://oness.sourceforge.net/proyecto/html/ch05s02.html>
- ftp://190.5.199.3/jjurado/Mejora%20de%20Procesos/XP_6JAI.pdf
- <http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Presentacion-XP.pdf>

- http://elblogdelfrasco.blogspot.com/2008_07_01_archive.html
- <http://users.dsic.upv.es/asignaturas/facultad/lsi/ejemploxp/index.html>
- <http://www.ogis-ri.co.jp/otc/swec/process/am-res/am/artifacts/crcModel.html>
- Critica MA.pdf
- Xtremeprogramming.org
- http://synergyca.net/index.php?p=1_9_Metodologias

Arquitectura MVC y Tecnología a utilizar

- <http://en.juantxu.net/doku.php/jee/3?do=index>
- http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/rivera_1_a/capitulo2.pdf
- http://feeds.feedburner.com/librosweb_es
- <http://es.wikipedia.org/w/index.php?>
- <http://www.orasite.com/index.php?>
- <http://creativecommons.org/licenses/by-nc-sa/2.0/legalcode>
- <http://www.oracle.com/index.html>
- <http://es.debugmodeon.com/debate/cuales-son-las-ventajas-de-jdeveloper>
- http://www.programacion.com/articulo/integracion_de_jsf_spring_e_hibernate_para_crear_una_aplicacion_web_del_mundo_real_307/3
- Jsfpdf

MANUAL DE USUARIO DE SIGFAP

INTRODUCCIÓN

La Asociación de Profesores de la FICA cuenta con un sistema de gestión de Fondos, el cual permite realizar varias transacciones contables, obtener reportes entre otros.

INSTRUCCIONES DE ACCESO

Como primer paso el usuario debe ingresar al navegador de Internet y escribir la dirección:

<http://172.20.14.114::8888/sigfap/faces/index.jsp>.

Aparece la página inicial en la cual se requiere que ingrese el nombre de usuario y una contraseña. En este caso el nombre de usuario será el número de cédula del usuario. Una vez de acuerdo con los datos hacer clic en “Aceptar”.



Fig.1Página de inicio de SIGFAP.

Al sistema SIGFAP se puede acceder como los siguientes usuarios:

- Administrador.
- Operador.
- Auditor.
- Socio.

De acuerdo al tipo de usuario que se haya autenticado ingresará al menú de acciones que puede realizar.

Accesos al Sistema como Administrador.

Una vez que el usuario de tipo Administrador se haya autenticado, tiene disponibles las siguientes opciones:

- Plan de Cuentas.
- Asientos Contables.
- Préstamos.
- Pagos.
- Reportes.
- Cierre de Periodos Contables.
- Historiales.
- Actualización de Datos.

Se muestra una página de Bienvenida en la cual se encuentra información de la Asociación de Profesores de la FICA.

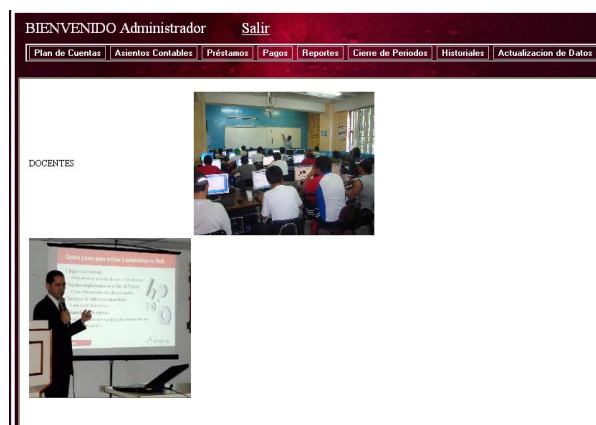


Fig.2Página inicial de la Autenticación de Administrador.

Si desea realizar alguna acción debe hacer clic sobre alguno de las opciones disponibles.

Plan de Cuentas

Si desea realizar alguna acción del plan de cuentas, debe hacer clic sobre la opción y se desplegará un menú:

- Reporte de Plan de Cuentas.
- Nueva Cuenta contable.



Fig.3Menú de opciones de "Plan de Cuentas"

Si escoge la opción “Plan de Cuentas” se despliega las cuentas Contables correspondientes al Plan de Cuentas de la Asociación de Profesores de la FICA.

Reporte de cuentas contables del Plan de Cuentas de la Asociación de la FICA.

Si escoge la opción “Nueva Cuenta Contable”, ingresará una cuenta contable para el Plan de Cuentas.



Fig.4 Opción “Nueva Cuenta Contable”.

Para ingresar una nueva cuenta contable, lo primero que debe escribir es el Identificador de la Cuenta contable (de acuerdo al Plan de cuentas establecido).

A screenshot of a web form titled 'Ingrese los Datos del Plan de Cuentas'. The form contains the following fields: 'Id Cuenta Contable' with the value '5201', 'Tipo de Cuenta' with a dropdown menu showing 'ACTIVO', 'Cuenta Contable Padre' with a dropdown menu showing 'No tiene Cuenta Padre', and 'Nombre' with an empty text box. A 'Guardar' button is located at the bottom.

Fig.5 Ingreso de datos del plan de Cuentas.

En el “Tipo de Cuenta” se despliega una listado con las opciones:

- Activo
- Pasivo
- Patrimonio
- Gasto
- Ingreso

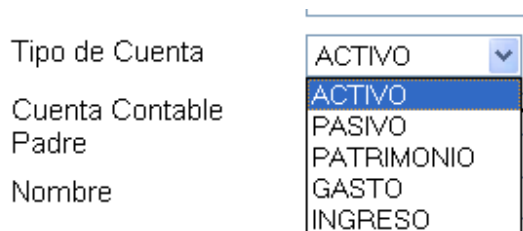


Fig.6 Opciones de “Tipo de Cuenta”.

La cuenta para la “Cuenta Contable Padre” es la correspondiente a la cuenta de la cual se deriva la cuenta contable que está ingresando. Y si es una cuenta contable de la cual se van a derivar otras cuentas, la opción que debe escoger es “No tiene Cuenta Padre”.

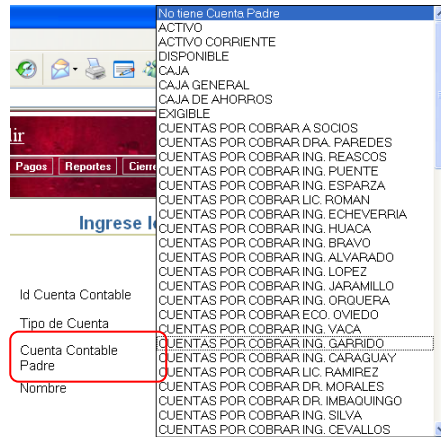


Fig.7 Listado de Cuentas para la “Cuenta Contable Padre”.

Y por último debe ingresar el nombre de la Cuenta Contable. Una vez ingresados todos los datos de la cuenta contable presionar el botón “Guardar”.

Y se muestra una ventana en la cual se ve un mensaje de que la cuenta contable se ha guardado exitosamente.

Asientos Contables

Si el Administrador escoge el menú “Asientos Contables” tiene dos opciones:

- Reportes.
- Nuevo Asiento Contable.



Fig.8 Opciones del menú Asientos Contables.

Si escoge la Opción “Reportes” del menú “Asientos Contables”, se muestra un listado de todos los asientos Contables que se han realizado en el Periodo contable activo.

Id	Periodo	Fecha	Concepto	Debe	Haber
5	4	31/12/1989	Valores estado de Situación Inicial	19702.89	19702.89
10	4	13/01/2010	pago 7 cuota Dr. Imbaquingo	72.0	72.0
15	4	15/01/2010	Aportes, Enero 2010 ctas por cobrar, interes	1932.0	1932.0
20	4	22/01/2010	Pago crédito ing. López	436.67	436.67
25	4	22/01/2010	Renovación crédito ing. López	1000.0	1000.0
30	4	29/01/2010	Aportes Enero 2010	446.0	446.0
35	4	29/01/2010	aportes Enero 2010	1486.0	1486.0
40	4	04/02/2010	Pago de crédito ing. Vaca	1706.55	1706.55
45	4	04/02/2010	Renovación crédito ing. Vaca	2500.0	2500.0
50	4	12/02/2010	Crédito ing. Caraguay	1500.0	1500.0
55	4	15/02/2010	Aportes febrero 2010, ctas por cobrar	1976.34	1976.34
60	4	19/02/2010	Compra de 6 botellas Grants Semana Curricular	78.0	78.0
70	4	22/02/2010	Cancelación de crédito ing. Caraguay	1505.0	1505.0
85	4	22/02/2010	Aporte de 8 profesores para Jornadas Curricular	64.0	64.0
76	4	25/02/2010	Pago cena Jornadas Curriculares	357.0	357.0
85	4	26/02/2010	Aportes, Febrero 2010	1530.34	1530.34

Fig.9 Listado de los Asientos Contables – Libro Diario.

En cada uno de los asientos Contables tiene dos enlaces: Detalle y Editar.

Si escoge la opción “Detalle” se muestra el Asiento contable detalladamente, cada uno de las cuentas contables que están involucradas en el asiento con su respectivo valor sea en el Debe o Haber.

DETALLE DE ASIENTO CONTABLE

Id Asiento Contable 10
 Concepto pago 7 cuota Dr. Imbaquingo
 Fecha 2010-01-13

Debe	Haber
72.0	72.0

Tipo Cuenta	Cuenta	Nombre	Debe	Haber
1	1110102	CAJA DE AHORROS	72.0	0.0
5	511	INTERESES GANADOS	0.0	4.98
1	1120120	CUENTAS POR COBRAR DR. IMBAQUINGO	0.0	67.02

Fig.10 Detalle del Asiento contable seleccionado.

En la Parte superior derecha hay un botón “Imprimir”, al seleccionarlo se muestra una ventana con los detalles del Asiento contable en formato pdf, el cual luego lo puede guardar o imprimir.

ASOCIACION DE PROFESORES DE LA FICA
 Asiento Contable

Id Asiento 10 Fecha 1/13/10 12:00
 Concepto pago 7 cuota Dr. Imbaquingo

Debe	Haber
72	72

id.Libro	CuentaContable	Nombre de Cuenta	Debe	Haber
10	11102	CAJA DE AHORROS	72	0
10	511	INTERESES GANADOS	0	4.98
10	1120120	CUENTAS POR COBRAR DR.	0	67.02

Fig.11 Detalles de un Asiento Contable en formato pdf.

Si selecciona la opción “Editar”, puede modificar algunos valores del Asiento Contable tales como: Periodo Contable, Concepto y Fecha. Una vez modificado alguno de estos valores presionar el botón “Guardar”.

BIENVENIDO Administrador Salir

Plan de Cuentas | Asientos Contables | Préstamos | Pagos | Reportes | Cierre de Periodos | Historiales | Actualización de Datos

DATOS DE LIBRO

Id Libro Diario: 5

Id Periodo Contable: 4

Concepto: Valores estado de Situ

Fecha: 01/01/2010

Debe: 19702.88

Haber: 19702.89

Guardar

Fig.12 Edición de los datos de un Asiento Contable.

NUEVO ASIEN TO

Si escoge la opción “Nuevo Asiento” del menú “Asientos Contables”, puede ingresar los datos correspondientes a un nuevo Asiento:

Primero en la cabecera del asiento contable, ingresará el concepto del asiento y la fecha. Al escoger la fecha hay un botón en la derecha el cual se despliega un calendario para escoger la fecha.

Los valores del Debe y Haber, se actualizarán dependiendo de los detalles que ingrese.

INGRESE LOS DATOS DEL LIBRO DIARIO

Concepto: Compra de Materiales

Fecha: 11/11/2011

Debe: 0

Haber: 0

Fig.13 Datos ingresados de la cabecera del nuevo Asiento contable.

Para ingresar los datos del Detalle del asiento contable debe primero escoger la cuenta contable de un listado:

ACTIVO CORRIENTE
DISPONIBLE
CAJA
CAJA GENERAL
CAJA DE AHORROS
EXIGIBLE
CUENTAS POR COBRAR A SOCIOS
CUENTAS POR COBRAR DPA. PAREDES
CUENTAS POR COBRAR ING. REASCOS
CUENTAS POR COBRAR ING. PUENTE
CUENTAS POR COBRAR ING. ESPARZA
CUENTAS POR COBRAR LIC. ROMAN
CUENTAS POR COBRAR ING. ECHEVERRIA
CUENTAS POR COBRAR ING. HUACA
CUENTAS POR COBRAR ING. BRAVO
CUENTAS POR COBRAR ING. ALVARADO
CUENTAS POR COBRAR ING. LOPEZ
CUENTAS POR COBRAR ING. JARAMILLO
CUENTAS POR COBRAR ING. ORQUERA
CUENTAS POR COBRAR ECO. OVIEDO
CUENTAS POR COBRAR ING. VACA
CUENTAS POR COBRAR ING. GARRIDO
CUENTAS POR COBRAR ING. CARAGUAY
CUENTAS POR COBRAR LIC. PAMPEZ
CUENTAS POR COBRAR DR. MORALES
CUENTAS POR COBRAR DR. IMBAQUINGO
CUENTAS POR COBRAR ING. SILVA
CUENTAS POR COBRAR ING. CEVALLOS
CUENTAS POR COBRAR ING. GRANDA

Ingresar Detalle Guardar

Id Cuenta	Nombre	Debe	Haber
-----------	--------	------	-------

Fig.14 Lista de cuentas contables.

Ingresar un valor ya sea en el Debe o en el Haber. No debe ingresar valores en los dos opciones en el Debe y Haber. Si lo hace se muestra un mensaje indicando que no es una acción correcta.

INGRESO DEL DETALLE

Id Cuenta Contable	Debe	Haber
CAJA GENERAL	40	5

* Los valores del detalle estan mal ingresados

Fig.15 Mensaje de ingreso de valores incorrectos.

Una vez de acuerdo con los datos ingresados hacer clic en el botón “Ingresar detalle”, y se muestra en la parte inferior en un listado la cuenta contable seleccionada con su valor correspondiente al Debe o Haber, y se actualiza el valor del Debe o Haber de la Cabecera del Asiento Contable.

INGRESO DEL DETALLE

Id Cuenta Contable	Debe	Haber
ACTIVO	0	0

Ingresar Detalle Guardar

Id	Cuenta	Nombre	Debe	Haber	
1	1110101	CAJA GENERAL	40.0	0.0	Eliminar

Fig.16 Ingreso de Detalle

Puede seguir ingresando los valores del Detalle del Asiento Contable repitiendo el mismo proceso. Una vez que el valor del Debe y Haber de la Cabecera del Asiento Contable sea igual, puede guardar el asiento.

The screenshot shows a web interface for entering journal entries. At the top, it says "INGRESE LOS DATOS DEL LIBRO DIARIO". Below this, there are fields for "Concepto" (Compra de Materiales) and "Fecha" (11/11/2011). Underneath, it displays "Debe" (40) and "Haber" (40). A section titled "INGRESO DEL DETALLE" contains a dropdown menu for "Id Cuenta Contable" (ACTIVO) and two input fields for "Debe" and "Haber", both containing the value 0. Below these are buttons for "Ingresar Detalle" and "Guardar". At the bottom, a table lists the details of the entry:

Id Cuenta	Nombre	Debe	Haber	
1	1110101 CAJA GENERAL	40.0	0.0	Eliminar
2	411 GASTOS VARIOS	0.0	40.0	Eliminar

Fig.17 Valores de Debe y Haber son Iguales

Si los valores son diferentes al momento de hacer clic en el botón “Guardar”, se muestra un mensaje en el que indica que los valores del Debe y Haber no coinciden.

The screenshot shows the same interface as Fig.17, but with "Debe" (43) and "Haber" (40) displayed. Below the "INGRESO DEL DETALLE" section, a message reads: "Los valores del DEBE y HABER del Asiento no son iguales". The "Id Cuenta Contable" dropdown is set to "ACTIVO" and the "Debe" and "Haber" input fields are both empty.

Fig.18 Mensaje de no coincidencia del Debe y Haber.

Préstamos

Si escoge el menú préstamos tiene dos opciones:

- Reportes.
- Nuevo Préstamo.



Fig.19 Menú Préstamos.

Si escoge la opción “Reportes” del menú “préstamos”, se despliega un listado de todos los préstamos que se encuentran activos.

DATOS DE PRÉSTAMO								
Id Amortizacion	Socio	Cuenta Contable	Fecha	Monto	Gracia	Amortizacion	Piizo	Nro Periodos
130	1702636190	1120110	2011-09-30	2500.0	0.0	30	2.0	24
140	0400600904	1120113	2011-09-30	2500.0	0.0	30	1.0	12
120	1001106119	1120103	2011-09-13	2500.0	0.0	30	2.0	24
135	1000982882	1120112	2011-09-30	2500.0	0.0	30	2.0	24
145	1000878472	1120131	2011-09-23	1000.0	0.0	30	1.0	24
150	1703695245	1120114	2011-07-25	2500.0	0.0	30	2.0	24
125	1000074764	1120122	2011-10-26	2500.0	0.0	30	2.0	24
100	1000959005	1120121	2011-03-04	560.82	0.0	30	1.0	12
105	0400454054	1120101	2011-03-04	1264.14	0.0	30	1.0	12
110	1102451687	1120117	2011-03-18	1000.0	0.0	30	1.0	12
5	1001545142	1120111	2009-08-05	2000.0	0.0	30	2.0	24
10	1703671501	1120105	2010-05-19	2000.0	0.0	30	2.0	24
30	1001701190	1120123	2010-07-07	600.0	0.0	30	1.0	12
40	1704725033	1120124	2010-09-27	2554.44	0.0	30	2.0	24
60	1001354701	1120109	2010-10-06	2500.0	0.0	30	2.0	24
80	1000400430	1120125	2010-12-02	1000.0	0.0	30	1.0	12
85	1000872109	1120108	2011-01-17	2500.0	0.0	30	2.0	24

Fig.20 Listado de datos de los préstamos vigentes.

En cada uno de los datos correspondientes a un Préstamo tiene una opción “Detalle”, el cual al hacer clic sobre esta opción se muestra una nueva ventana con los datos detallados del Préstamo.

BIENVENIDO Administrador Salir							
Plan de Cuentas Asientos Contables Préstamos Pagos Reportes Cierre de Periodos Historiales Actualización de Datos							
DETALLE DE AMORTIZACION							
Imprimir							
Id Amortizacion 130 Id Socio 1702636190 Nombre LOPEZ GUILLERMO id Cuenta 1120110 Monto 2500.0 Nro Periodos 24							
Id Detalle	Vencimiento	Valor Interes	Principal	Dividendo	Saldo	Pagada	
469	2011-10-30	31.25	88.97	121.22	2410.03	1	
470	2011-11-29	30.13	91.09	121.22	2318.94	1	
471	2011-12-29	28.99	92.23	121.22	2226.71	0	
472	2012-01-28	27.83	93.39	121.22	2133.32	0	
473	2012-02-27	26.67	94.55	121.22	2038.77	0	
474	2012-03-28	25.48	95.74	121.22	1943.03	0	
475	2012-04-27	24.29	96.93	121.22	1846.1	0	
476	2012-05-27	23.08	98.14	121.22	1747.96	0	
477	2012-06-26	21.85	99.37	121.22	1648.59	0	

Fig.21 Datos detallados de un Préstamo vigente.

Si desea imprimir los datos del préstamo, en la parte superior derecha existe un botón “Imprimir”, que al presionarlo se muestra los datos de quien solicitó y la tabla de amortización en formato pdf.

Si desea puede guardar esta información o bien la puede imprimir en papel.

FECHA DE VENCIMIENTO	DIVIDENDO	PRINCIPAL	VALOR INTERES	SALDO
06/30/11 12:00 AM	121.22	89.97	31.25	2410.03
11/30/11 12:00 AM	121.22	91.09	32.13	2318.84
12/30/11 12:00 AM	121.22	92.33	32.59	2226.71
1/28/12 12:00 AM	121.22	93.39	37.83	2133.32
2/27/12 12:00 AM	121.22	94.88	36.87	2038.77
3/28/12 12:00 AM	121.22	95.74	35.48	1943.03
4/27/12 12:00 AM	121.22	96.93	34.29	1846.1
5/27/12 12:00 AM	121.22	98.14	33.08	1747.96
6/26/12 12:00 AM	121.22	99.37	31.86	1648.59
7/26/12 12:00 AM	121.22	100.61	30.61	1547.98
8/25/12 12:00 AM	121.22	101.87	29.35	1446.11
9/24/12 12:00 AM	121.22	103.14	28.08	1342.97
10/24/12 12:00 AM	121.22	104.43	26.79	1238.54
11/23/12 12:00 AM	121.22	105.74	25.48	1132.8
12/23/12 12:00 AM	121.22	107.06	24.15	1025.74

Fig.22 Datos de un préstamo en formato pdf.

Si escoge la opción “Nuevo Préstamo” del menú “Préstamos”, le da la opción de ingresar los datos para un nuevo préstamo.



Fig.23 Menú Préstamos – Nuevo Préstamo

Al escoger la opción de “Nuevo Préstamo” se muestra la ventana de ingreso de datos.

Fig.24 Ingreso de Datos de un nuevo Préstamo.

El primer dato que debe ingresar para un nuevo préstamo es escoger la cuenta contable de un listado de cuentas correspondientes a los socios.

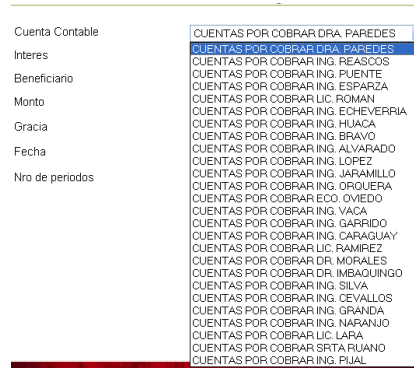


Fig.25 Cuentas Contables de los socios.

Luego debe escoger el valor del Interés a aplicar a la tabla de amortización para el préstamo.



Fig.26 Valor de Interés.

Luego debe escoger el Nombre del Socio quien solicita el préstamo.

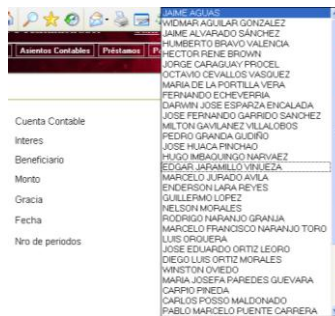


Fig.27 Listado de los nombres de Socios.

Además, debe ingresar los datos correspondientes al monto, si es mayor al permitido se mostrará un mensaje.

Debe ingresar el Plazo, el valor de Gracia y la Amortización cada cuantos días.

Seleccionar la fecha de un calendario desplegable. Y finalmente debe ingresar el número de cuotas.

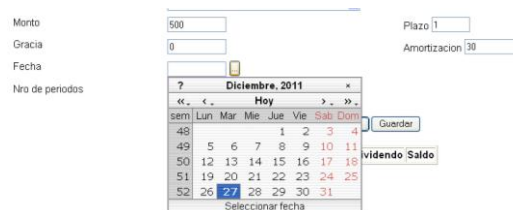


Fig.28 Datos de Préstamo.

Una vez de acuerdo con los datos proporcionados, hacer clic en “Generar Amortización”, se despliega la tabla de amortización correspondiente.

Ingrese los datos del Préstamo

Cuenta Contable: CUENTAS POR COBRAR ING. ORQUERA
 Interes: 0.15
 Beneficiario: LUIS ORQUERA
 Monto: 2500 Plazo: 2
 Gracia: 0 Amortizacion: 30
 Fecha: 06/10/2011
 Nro de periodos: 24

[Generar Amortizacion] [Guardar]

Vencido	Principal	Interes	Dividendo	Saldo
05/11/2011	89.87	31.25	121.22	2410.03
05/12/2011	81.09	30.13	121.22	2318.84
04/01/2012	92.23	28.99	121.22	2226.71
03/02/2012	93.39	27.83	121.22	2133.32
04/03/2012	94.55	26.67	121.22	2038.77
03/04/2012	95.74	25.48	121.22	1943.03

Fig.29 Tabla de amortización del préstamo solicitado.

Si esta de acuerdo con los datos proporcionados haga clic en “Guardar”.

Si los datos no son los correctos puede ingresar valores diferentes y nuevamente debe generar la tabla de amortización y luego hacer clic en “Guardar”.

En el caso de que el socio tenga un préstamo vigente no podrá guardar un nuevo préstamo, si es así se muestra un mensaje indicando que: “El socio debe cancelar el crédito que tiene”.

27/07/2013	116.79	443.0	121.22	23790.0
26/08/2013	118.25	297.0	121.22	11965.0
25/09/2013	119.72	150.0	121.22	-7.0

• El socio debe cancelar el crédito que tiene

Fig.30 Mensaje: “El socio debe cancelar el crédito que tiene”.

Pagos

Si escoge la opción “Pagos”, tiene tres posibilidades:

- Pago de crédito.
- Pago de Interés.
- Aportaciones y cuotas mensuales.



Fig.31 Menú “Pagos”.

Si escoge la opción “Pago de Crédito”, se muestra una ventana en la cual un socio puede cancelar el crédito.

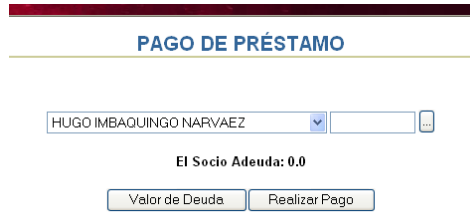


Fig.32 Pago de Préstamo.

Se encuentra un listado de todos los socios, debe escoger el nombre del socio que desea cancelar el crédito.

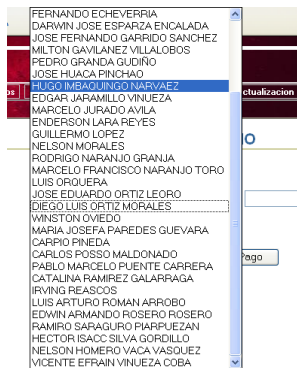


Fig.33 Listado de socios que cancelarán el crédito.

Además debe ingresar la fecha en la cual hace el pago del crédito para ello debe hacer clic en el botón “...” y se despliega el calendario para escoger la fecha.



Fig.34 Fecha de pago.

Una vez escogido el nombre del socio, y la fecha de pago puede revisar el valor que adeuda el haciendo clic en el botón “Valor de Deuda”, y en la parte inferior se muestra el mensaje: “El socio Adeuda: ...” en este caso se indica 0,0 significa que al momento no tiene ningún préstamo.

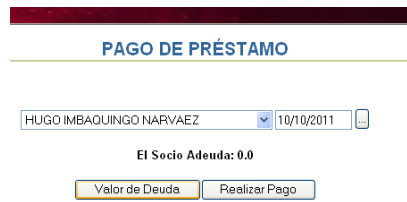


Fig.35 Valor que el Socio Adeuda 0,0.

De igual forma si decide Realizar el pago del crédito haciendo clic en “Realizar pago”, como no tenía ningún crédito se muestra un mensaje indicando que el socio no tiene ningún préstamo.

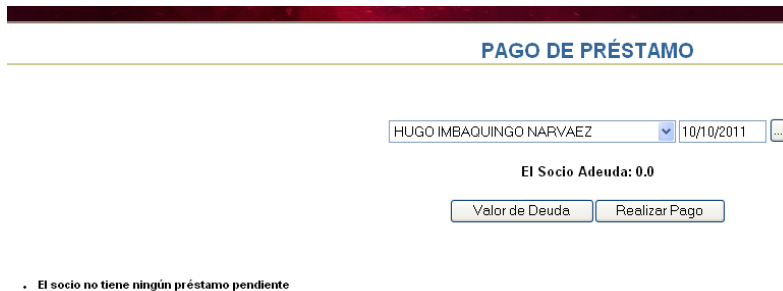


Fig.36 Mensaje: El socio no tiene ningún préstamo pendiente

Ahora si escoge el nombre de otro socio, al hacer clic en el botón “Valor de Deuda” se muestra el valor en: El socio Adeuda:... , en esta ocasión si sale un valor diferente de 0,0.



Fig.37 Valor Adeudado por el socio diferente de 0.0

Una vez consultado el valor que adeuda el socio debe hacer clic en el botón “Realizar Pago”. Se realiza el pago y se queda registrado en el libro diario.

Si escoge la opción “Pago de Interés” del menú “Pagos”



Fig.38 Opción “Pago de Interés”

Se muestra una ventana en la cual tiene dos botones “Lista Pago” y “Pagar”.



Fig.39 Ventana Pago de Interés.

Al hacer clic en el botón “Lista Pago”, se despliega un listado de todos los socios con su respectiva cuenta contable, el aporte realizado a la Asociación y el valor de la ganancia del interés que le correspondería.

Idsocio	Nombre	Cuenta	Nombre Cuenta	Aporte	Ganancia
0400454054	PAREDES GUEVARA MARIA JOSEFA	1120101	CUENTAS POR COBRAR DRA. PAREDES	780.0	267.9728712059065
0400600904	OVIEDO WINSTON	1120113	CUENTAS POR COBRAR ECO. OVIEDO	780.0	267.9728712059065
0400995437	RAMIREZ GALARRAGA CATALINA	1120118	CUENTAS POR COBRAR LIC. RAMIREZ	550.0	188.95522969647264
0600817290	VINUEZA COBA VICENTE EFRAIN	1120127	CUENTAS POR COBRAR DR. VINUEZA	780.0	267.9728712059065
0701033119	AGUILAR GONZALEZ WIDMAR	1120128	CUENTAS POR COBRAR ING. AGUILAR	490.0	164.90638228055786
1000074764	CEVALLOS VASQUEZ OCTAVIO	1120122	CUENTAS POR COBRAR ING. CEVALLOS	780.0	267.9728712059065
1000400430	LARA REYES ENDERSON	1120125	CUENTAS POR COBRAR LIC. LARA	780.0	267.9728712059065
1000596708	ORTIZ LEORO JOSE EDUARDO	1120129	CUENTAS POR COBRAR ING. ORTIZ L.	780.0	267.9728712059065
1000706133	POSSO MALDONADO CARLOS	1120130	CUENTAS POR COBRAR ARQ. POSSO	780.0	267.9728712059065
1000785188	EACHEVERRIA FERNANDO	1120106	CUENTAS POR COBRAR ING. EACHEVERRIA	780.0	267.9728712059065
1000811552	IMBAQUINGO NARVAEZ HUGO	1120120	CUENTAS POR COBRAR DR. IMBAQUINGO	780.0	267.9728712059065
1000872109	BRAVO VALENCIA HUMBERTO	1120108	CUENTAS POR COBRAR ING. BRAVO	490.0	164.90638228055786
1000978479	AGUILAR WIDMAR	1120131	CUENTAS POR COBRAR ING. AGUILAR	490.0	164.90638228055786

Fig.40 Valor de Ganancia de Interés por aportes.

Al hacer clic en el botón “Pagar” se realiza el pago a cada uno de los socios el valor correspondiente de ganancia, y si vuelve consultar saldrá cero.



Fig.41 Pagar crédito.

Al escoger la opción “Aportaciones y cuotas mensuales” del menú “Pagos”.



Fig.42 Opción “Aportaciones y cuotas mensuales”

Se muestra una ventana con un listado de todos los socios y los aportes correspondientes a: Aporte Asociación, Aporte Caja y Aporte FDU.

APORTES DE LOS SOCIOS

Id Socio	Nombre	Apellido	Aporte Aso	Aporte Caja	Aporte Fdu
1000876472	JAIME	AGUAS	10.0	10.0	2.0
0701033119	INDMAR	AGUILAR GONZALEZ	10.0	10.0	2.0
1001354701	JAIME	ALVARADO SANCHEZ	10.0	10.0	2.0
1000872109	HUMBERTO	BRAVO VALENCIA	10.0	10.0	2.0
1000983756	HECTOR RENE	BROWN	10.0	10.0	2.0
1102451687	JORGE	CARAQUAY PROCEL	10.0	10.0	2.0
1000074784	OCTAVIO	CEVALLOS VASQUEZ	10.0	10.0	2.0
1001133063	MARIA	DE LA PORTILLA VERA	10.0	10.0	2.0
1000765188	FERNANDO	ECHEVERRIA	10.0	10.0	2.0
1001584570	DARWIN JOSE	ESPARZA ENCALADA	10.0	10.0	2.0
1707852081	JOSE FERNANDO	GARRIDO SANCHEZ	10.0	10.0	2.0
1001139458	MILTON	GAVILANEZ VILLALOBOS	10.0	10.0	2.0
1001701190	PEDRO	GRANDA GUDINO	10.0	10.0	2.0
1000976421	JOSE	HUACA PINCHAO	10.0	10.0	2.0

Fig.43 Ventana de aportes de socios.

Inicialmente el aporte para todos los socios es el mismo. Si algún socio aporta un valor diferente tiene la posibilidad de editar ese valor.

100011552	MIGUEL	IBARRA JINJUN NARYA EZ	10.0	10.0	2.0
1001545142	EDGAR	JARAMILLO VINUEZA	10.0	10.0	2.0
1001303682	MARCELO	JURADO AVILA	10.0	0	2.0
1000400430	ENDERSON	LARA REYES	10.0	10.0	2.0
1702636190	GUILLEMO	LOPEZ	10.0	10.0	2.0
1701298174	NELSON	MORALES	10.0	10.0	
1704725033	RODRIGO	NARANJO GRANJA	10.0	10.0	2.0
1706870464	MARCELO FRANCISCO	NARANJO TORO	10.0	10.0	2.0
1000982882	LUIS	ORQUERA	10.0	10.0	2.0
1000596708	JOSE EDUARDO	ORTIZ LEORO	10.0	10.0	2.0
1001586997	DIEGO LUIS	ORTIZ MORALES	10.0	10.0	2.0

Fig.44 Edición de valores de aportes.

Una vez de acuerdo con todos los valores de aportes de los socios debe hacer clic en el botón “Aceptar” y se guardan los aportes de los socios.

VINUEZA COBA	10.0	10.0
<input type="button" value="Aceptar"/>		

Fig.45 Botón Aceptar

Se muestra una nueva ventana en la cual se mostrarán todos los socios que están adeudando a la fecha que ingresemos.

Socios que adeudan a la fecha

Nombre Cuenta	Apellido Socio	Nombre Socio	Principal	Interes	Pagar
Aportes Caja			Aportes Asociación		
<input type="text" value="370.0"/>			<input type="text" value="380.0"/>		
Aportes FDU			<input type="text" value="66.0"/>		

Fig.46 Ventana de pago de cuotas adeudadas a la fecha.

Debe ingresar la fecha en la cual se consideran todos los socios que adeudan desde esa fecha más 30 días.

12/12/2011

? Diciembre, 2011

Diciembre, 2011						
Hoy						
sem	Lun	Mar	Mie	Jue	Vie	Sab Dom
48				1	2	3 4
49	5	6	7	8	9	10 11
50	12	13	14	15	16	17 18
51	19	20	21	22	23	24 25
52	26	27	28	29	30	31

Lun, Dic 12

Fig.47 Fecha a partir de la cual se consideran los socios que adeudan.

Una vez ingresada la fecha debe hacer clic en el botón “Aceptar” y se muestra el listado de los socios deudores. Debe ir seleccionando los socios quienes pagarán la cuota en ese mes.

Nombre Cuenta	Apellido Socio	Nombre Socio	Principal	Interes	Pagar
CUENTAS POR COBRAR ING. AGUAS	AGUAS	JAIME	79.72	10.54	<input checked="" type="checkbox"/>
CUENTAS POR COBRAR ING. ALVARADO	ALVARADO SÁNCHEZ	JAIME	107.06	14.16	<input checked="" type="checkbox"/>
CUENTAS POR COBRAR ING. BRAVO	BRAVO VALENCIA	HUMBERTO	101.87	19.35	<input checked="" type="checkbox"/>
CUENTAS POR COBRAR ING. CARAGUAY	CARAGUAY PROCEL	JORGE	85.88	4.38	<input checked="" type="checkbox"/>
CUENTAS POR COBRAR ING. CEVALLOS	CEVALLOS VASQUEZ	OCTAVIO	91.09	30.13	<input checked="" type="checkbox"/>
CUENTAS POR COBRAR ING. LOPEZ	LOPEZ	GUILLERMO	92.23	28.99	<input checked="" type="checkbox"/>
CUENTAS POR COBRAR ING. NARANJO	NARANJO GRANJA	RODRIGO	110.76	13.1	<input checked="" type="checkbox"/>
CUENTAS POR COBRAR ING. ORQUERA	ORQUERA	LUIS	92.23	28.99	<input checked="" type="checkbox"/>
CUENTAS POR COBRAR ECO. OMIEDO	OMIEDO	WINSTON	199.29	26.36	<input type="checkbox"/>
CUENTAS POR COBRAR DRA. PAREDES	PAREDES GUEVARA	MARIA JOSEFA	109.93	4.17	<input checked="" type="checkbox"/>
CUENTAS POR COBRAR ING. PUENTE	PUENTE CARRERA	PABLO MARCELO	92.23	28.99	<input checked="" type="checkbox"/>
CUENTAS POR COBRAR LIC. ROMAN	ROMAN ARROBO	LUIS ARTURO	91.13	5.84	<input checked="" type="checkbox"/>
CUENTAS POR COBRAR ING. SILVA	SILVA GORDILLO	HECTOR ISACC	48.77	1.85	<input checked="" type="checkbox"/>
CUENTAS POR COBRAR ING. VACA	VACA VASQUEZ	NELSON HOMERO	94.55	26.67	<input checked="" type="checkbox"/>

Fig.48 Lista de socios deudores.

Cuando esté de acuerdo con los socios que cancelan la cuota debe hacer clic en el botón “Guardar”

Fig.49 Guardar los pagos de los socios.

Reportes

Si escoge el menú “Reportes” tiene las siguientes opciones:

- Estado de cuenta del Socio.
- Mayorización
- Balance de cuentas.
- Reporte Aportes.
- Pérdidas y Ganancias Caja General.
- Pérdidas y Ganancias Caja de Ahorros.
- Estado de Situación Final.
- Estado de Situación Financiera Caja de Ahorros.
- Estado de Situación Financiera Caja General
- Aportaciones de Socios.

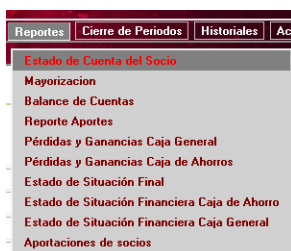


Fig.50 Opciones del menú “Reportes”.

Si escoge la opción “Estado de cuenta del Socio” del menú “Reportes”. Se despliega una ventana en la cual se muestra una opción para escoger el nombre de un socio.

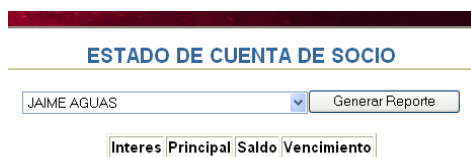


Fig.51 Ventana de Estado de Cuenta de Socio.

Al escoger el nombre del socio se despliega un listado de todos los socios.

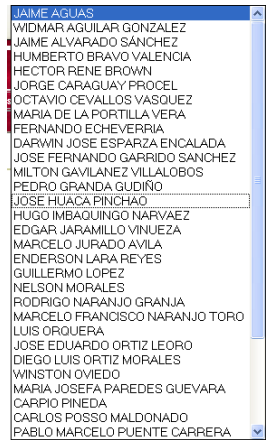


Fig.52 Lista de socios para Estado de Cuenta.

Al escoger un nombre y hacer clic en el botón “Generar Reporte”, se despliega un listado de todas las cuotas adeudadas por el socio.

ESTADO DE CUENTA DE SOCIO

JAIME AGUAS Generar Reporte

Interes	Principal	Saldo	Vencimiento
9.55	80.71	683.08	2012-01-21
8.54	81.72	601.36	2012-02-20
7.52	82.74	518.82	2012-03-20
6.48	83.78	434.84	2012-04-19
5.44	84.82	350.02	2012-05-19
4.38	85.88	264.14	2012-06-18
3.3	86.96	177.18	2012-07-18
2.21	88.05	89.13	2012-08-17
1.11	89.15	-0.02	2012-09-16

Fig.53 Cuotas adeudadas de un socio.

Si escoge la opción “Mayorización” del menú “Reportes”. Se despliega una ventana en la cual puede escoger la cuenta contable la cual quiere saber el movimiento financiero que ha tenido.

Al seleccionar una Cuenta Contable se despliega un listado con todas las Cuentas Contables del Plan de Cuentas.

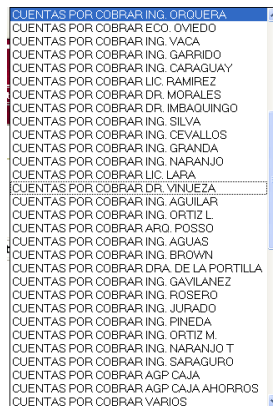


Fig.54 Lista de Cuentas Contables para mayorizar.

Al seleccionar una cuenta contable debe hacer clic en el botón “Mayorización” y se muestran todos los movimientos financieros que ha tenido la Cuenta Contable en el periodo Contable.

Si escoge la opción “Balance de Cuentas” del menú “Reportes”. Se va a desplegar todas las cuentas contables del Plan de Cuentas con los valores correspondientes al Saldo Debe, Saldo Haber, Total Debe y Total Haber.

BALANCE DE CUENTAS				
Nombre	Saldo Debe	Saldo Haber	Total Debe	Total Haber
DISPONIBLE	0.0	0.0	0.0	0.0
CAJA	0.0	0.0	0.0	0.0
CAJA GENERAL	11236.28	6822.65	4413.63	0.0
CAJA DE AHORROS	72116.7	64752.84	7363.86	0.0
EXIGIBLE	0.0	0.0	0.0	0.0
CUENTAS POR COBRAR A SOCIOS	0.0	0.0	0.0	0.0
CUENTAS POR COBRAR DRA. PAREDES	2613.5	2171.68	441.82	0.0
CUENTAS POR COBRAR ING. REASCOS	0.0	0.0	0.0	0.0
CUENTAS POR COBRAR ING. PUENTE	6554.44	4144.41	2410.03	0.0
CUENTAS POR COBRAR ING. ESPARZA	0.0	0.0	0.0	0.0
CUENTAS POR COBRAR LIC. ROMAN	2000.0	1353.88	646.12	0.0
CUENTAS POR COBRAR ING. ECHEVERRIA	0.0	0.0	0.0	0.0

Fig.55 Balance de Cuentas.

Al final del balance se puede observar los totales de cada uno de los valores.

INGRESO APORTE CAJA AHORRO	0.0	23690.0	0.0	23690.0
INGRESO APORTE FDU	1400.0	2567.29	0.0	1167.29
TOTAL	217630.16	217630.16	217630.16	217630.16

Fig.56 Total de cada uno de los valores.

Si escoge “Reporte de Aportes” del menú “Reportes” se despliegan las cuentas contables con sus valores sea del debe o haber dependiendo de la agrupación en la que se encuentre.

APORTE CAJA	
Ingresos	
INGRESO APORTE CAJA AHORRO	23690.0
INTERESES GANADOS	8375.87
TOTAL	32065.870000000003
Cuentas por cobrar	
Nombre	Total
CUENTAS POR COBRAR ING. PUAL	1943.05
TOTAL CUENTAS POR COBRAR	25507.26
TOTAL APORTE CAJA	6558.610000000004
<small>(Ingresos - Cuentas x cobrar)</small>	

Fig.57 Reporte de Aportes.

Al escoger la opción “Pérdidas y Ganancias Caja General”, se muestra los valores de las cuentas contables que se encuentran involucradas con la Caja General, determinando la Utilidad.

ESTADO DE PÉRDIDAS Y GANANCIAS CAJA GENERAL	
INGRESOS	9950.59
+ Aportes Socios	8783.3
+ Aportes FDU	1167.29
MENOS GASTOS	5422.85
- Gastos Varios	5422.85
= Utilidad	3360.6499999999996

Fig.58 Estado de Pérdidas y Ganancias Caja General.

Al escoger la Opción “Estado de Pérdidas y Ganancias de la Caja de Ahorros” del menú “Reportes”, se muestran todas las cuentas Contables que se encuentran relacionadas con la Caja de Ahorros con su respectivo valor, de acuerdo a esto se sabrá la Utilidad que se ha generado

ESTADO DE PÉRDIDAS Y GANANCIAS CAJA DE AHORROS

INGRESOS	
Ingreso Aporte Caja	23690.0
Ingreso Interes	8375.87
Total	32065.870000000003
GASTOS	
Total Gastos	0
UTILIDAD	
Utilidad	32065.870000000003

Fig.59 Estado de Pérdidas y Ganancias Caja de Ahorros

Al escoger la Opción “Estado de Situación Final” del menú “Reportes”, se muestran las cuentas contables con el valor sea del Debe o Haber, dependiendo de la agrupación en la que se encuentre es decir: Activos, Pasivos, Patrimonio, Gastos e Ingresos.

ESTADO DE SITUACION FINAL	
Activos	
DISPONIBLE	0.0
CAJA	0.0
CAJA GENERAL	4413.63
CAJA DE AHORROS	7363.86
EXIGIBLE	0.0
CUENTAS POR COBRAR A SOCIOS	0.0
CUENTAS POR COBRAR DRA. PAREDES	441.82
CUENTAS POR COBRAR ING. REASCOS	0.0
CUENTAS POR COBRAR ING. PUENTE	2410.03
CUENTAS POR COBRAR ING. ESPARZA	0.0
CUENTAS POR COBRAR LIC. ROMAN	846.12
CUENTAS POR COBRAR ING. ECHEVERRIA	0.0
CUENTAS POR COBRAR ING. HUACA	0.0
CUENTAS POR COBRAR ING. BRAVO	1646.61
CUENTAS POR COBRAR ING. ALVARADO	1238.88
CUENTAS POR COBRAR ING. LOPEZ	2336.26
CUENTAS POR COBRAR ING. JARAMILLO	0.01
CUENTAS POR COBRAR ING. ORQUERA	2409.95

Fig.60 Estado de Situación Final

Al escoger la opción “Estado de Situación Financiera de la Caja de Ahorro” del menú “Reportes”, se muestra los datos correspondientes a los Activos, Pasivos y Utilidad de las Cuentas relacionadas con la Caja de Ahorro.

ESTADO DE SITUACION FINANCIERA CAJA DE AHORRO	
ACTIVOS	
Caja de Ahorro	7363.86
Cuentas por cobrar	
Nombre	Total Debe
CUENTAS POR COBRAR ING. PUJAL	1943.05
total Cuentas por Cobrar	25507.26
Total Activos	32871.119999999995
PASIVOS	
Total Pasivos	0
PATRIMONIO	
Utilidad	32065.87000000000005

Fig.61 Estado de Situación Financiera de la Caja de Ahorro

Al escoger la opción “Estado de Situación Financiera de la Caja General” del menú “Reportes”, se muestra los datos correspondientes a los Activos, Pasivos y Utilidad de las Cuentas relacionadas con la Caja General.

ESTADO DE SITUACION FINANCIERA CAJA DE AHORRO	
ACTIVOS	
Caja de General	4413.63
PASIVOS	
Pasivos	0
PATRIMONIO	
Utilidad	4527.94000000000005

Fig.62 Estado de Situación Financiera de la Caja General

Al escoger “Aporte de los Socios” del menú “Reportes”, se despliega un listado de todos los socios con el aporte respectivo a: Asociación, Caja General, FDU.

APORTES DE LOS SOCIOS				
Cédula	Nombre	Aporte Caja	Aporte Asociacion	Aporte F D U
0400454054	PAREDES GUEVARA MARIA JOSEFA	790.0	484.0	94.0
0400600804	OVIEDO WINSTON	790.0	484.0	94.0
0400985437	RAMIREZ GALARRAGA CATALINA	560.0	484.0	94.0
0600817290	VINUEZA COBA VICENTE EFRAIN	790.0	484.0	0.0
0701033119	AGUILAR GONZALEZ WIDMAR	490.0	484.0	94.0
1000074784	CEVALLOS VASQUEZ OCTAVIO	790.0	484.0	94.0
1000400430	LARA REYES ENDERSON	790.0	484.0	94.0
1000596708	ORTIZ LEORO JOSE EDUARDO	790.0	484.0	94.0
1000706133	POSSO MALDONADO CARLOS	790.0	484.0	2.0
1000785188	ECHEVERRIA FERNANDO	790.0	484.0	94.0
1000811552	IMBAQUINGO NARVAEZ HUGO	790.0	484.0	94.0
1000872109	IBRAVO VALENCIA HUMBERTO	490.0	484.0	94.0
1000876472	AGUAS JAIME	490.0	484.0	94.0
1000959005	SILVA GORDILLO HECTOR ISACC	490.0	484.0	94.0
1000978421	HUACA PINCHAO JOSE	790.0	484.0	94.0
1000982882	ORQUERA LUIS	790.0	484.0	94.0

Fig.63 Aporte de los Socios

Cierre de periodos contables

El menú “Cierre de periodos” consta de las siguientes opciones:

- Realizar Cierre.
- Informes de cierres.



Fig.64 Opciones del menú “Cierre de Periodos”

Al escoger la opción “Realizar Cierre” del menú “Cierre de Periodos” se muestra una ventana en la cual debe escoger la fecha en la cual se hace el cierre de periodo contable, luego debe hacer clic en el botón “Cerrar Periodo”.



Fig.65 Cierre de Periodo Contable.

Al escoger la opción "Informes de Cierres" del menú “Cierre de Periodo”, se va a visualizar un reporte de las fechas en las cuales se ha realizado un cierre de Periodo contable.

Historiales

El menú historiales tiene las siguientes opciones:

- Historial Préstamos.
- Historial Estado situación Final.



Fig.66 Opciones del menú “Historiales”.

Si escoge la opción “Historial de Préstamos” se despliega un listado de los socios que tienen un préstamo con los datos correspondientes.

DATOS DE PRÉSTAMO									
Id Amortizacion	Socio	Cuenta Contable	Fecha	Monto	Gracia	Amortizacion	Plazo	Nro Periodos	
130	1702636190	1120110	2011-09-30	2500.0	0.0	30	2.0	24	Detalle
140	0400600904	1120113	2011-09-30	2500.0	0.0	30	1.0	12	Detalle
120	1001106119	1120103	2011-09-13	2500.0	0.0	30	2.0	24	Detalle
135	1000982892	1120112	2011-09-30	2500.0	0.0	30	2.0	24	Detalle
145	1000976472	1120131	2011-09-23	1000.0	0.0	30	1.0	24	Detalle
150	1703695245	1120114	2011-07-25	2500.0	0.0	30	2.0	24	Detalle
125	1000074764	1120122	2011-10-26	2500.0	0.0	30	2.0	24	Detalle
100	1000959005	1120121	2011-03-04	560.82	0.0	30	1.0	12	Detalle
105	0400454054	1120101	2011-03-04	1264.14	0.0	30	1.0	12	Detalle
110	1102451687	1120117	2011-03-18	1000.0	0.0	30	1.0	12	Detalle
5	1001545142	1120111	2009-08-05	2000.0	0.0	30	2.0	24	Detalle
10	1703671501	1120105	2010-05-19	2000.0	0.0	30	2.0	24	Detalle
30	1001701190	1120123	2010-07-07	600.0	0.0	30	1.0	12	Detalle
40	1704725033	1120124	2010-08-27	2554.44	0.0	30	2.0	24	Detalle
60	1001354701	1120109	2010-10-08	2500.0	0.0	30	2.0	24	Detalle
80	1000400430	1120125	2010-12-02	1000.0	0.0	30	1.0	12	Detalle
85	1000872109	1120108	2011-01-17	2500.0	0.0	30	2.0	24	Detalle

Fig.67 Historial de Préstamos.

Además tiene la posibilidad de revisar los detalles del préstamo tiene que hacer clic en el enlace “Detalle”. Y si quiere imprimir la tabla de amortización generada debe hacer clic en el botón “Imprimir”.

DETALLE DE AMORTIZACION						
Imprimir						
Id Amortizacion 130						
Id Socio 1702636190						
Nombre LOPEZ GUILLERMO						
id Cuenta 1120110						
Monto 2500.0						
Nro Periodos 24						
Id Detalle	Vencimiento	Valor Interes	Principal	Dividendo	Saldo	Pagada
469	2011-10-30	31.25	99.97	121.22	2410.03	1
470	2011-11-29	30.13	91.09	121.22	2318.94	1
471	2011-12-28	28.99	82.23	121.22	2226.71	1
472	2012-01-28	27.83	73.39	121.22	2133.32	0
473	2012-02-27	26.67	64.55	121.22	2038.77	0
474	2012-03-28	25.48	55.74	121.22	1943.03	0
475	2012-04-27	24.29	46.93	121.22	1846.1	0
476	2012-05-27	23.08	38.14	121.22	1747.96	0
477	2012-06-26	21.85	29.37	121.22	1648.59	0

Fig.68 Detalles de una tabla de amortización.

Actualización de datos

El menú “Actualización de Datos” tiene la opción:

- Cambio de Contraseña.



Fig.69 Menú Actualización de Datos.

Al escoger “Cambio de Contraseña” del menú “Actualización de Datos”, le da la posibilidad de guardar una nueva contraseña con la cual ingresará al sistema. Una vez de acuerdo con los datos debe hacer clic en el botón “Aceptar”.

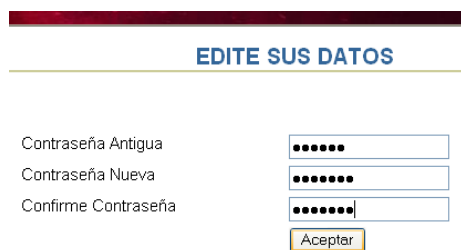
A screenshot of a web form titled 'EDITE SUS DATOS' in blue text. Below the title are three input fields: 'Contraseña Antigua' with 6 dots, 'Contraseña Nueva' with 6 dots, and 'Confirme Contraseña' with 6 dots. A yellow 'Aceptar' button is at the bottom right.

Fig.70 Cambio de Contraseña.

Acceso al Sistema como Operador

En la página de autenticación si el usuario y la contraseña es el correspondiente a un usuario de tipo “Operador” se mostrará el siguiente menú:

- Periodos Contables.
- Interés
- Tipo de Cuenta
- Parámetros
- Usuarios.

La página inicial es la de Bienvenida en la cual se muestran imágenes de los docentes.

Periodo contable

Al escoger el menú “Periodo Contable” tiene dos opciones:

- Reporte

- Nuevo Periodo.



Fig.71 Menú Periodo Contable.

Al escoger “Reporte” del menú “Periodo Contable” se despliega un listado de todos los periodos contables que se ha ingresado, mostrando las fechas de inicio y fin y si se encuentran activos o no.

PERIODOS CONTABLES				
Id Periodo Contable	Fecha Fin	Fecha Inicio	Estado	
6	2012-12-31	2012-01-02	1	Editar
2	2009-12-31	2009-01-01	0	Editar
4	2010-12-31	2010-01-01	1	Editar

Fig.72 Reporte de Periodos Contables.

Si desea cambiar algún valor del periodo contable debe hacer clic en el enlace “Editar” se muestra una nueva ventana en la cual se muestran los datos correspondientes al Periodo Contable seleccionado y nos da la posibilidad de cambiar: Fechas de Inicio y Fin.

EDICION DE PERIODO CONTABLE

Id Periodo Contable: 6

Fecha Inicio: 06/07/2007

Fecha Fin: [Calendar]

Fig.73 Edición de fechas de un Periodo Contable.

Una vez de acuerdo con los cambios realizados debe hacer clic en el botón “Guardar”.

EDICION DE PERIODO CONTABLE

Id Periodo Contable: 6

Fecha Inicio: 06/07/2007

Fecha Fin: 2012-12-31 00:00:00.0

Fig.74 Guardar cambios realizados a un Periodo Contable

Al escoger la opción “Nuevo Periodo” del menú “Periodo Contable”. Da la posibilidad de especificar las Fechas de inicio y Fin para el nuevo periodo, además de especificar si está activo o no. Una vez de acuerdo con los datos hacer clic en el botón “Guardar”.

Fig.75 Ingreso de Datos de un nuevo Periodo Contable.

Interés

En el menú “Interés” se encuentran estas opciones:

- Reportes
- Nuevo Interés.



Fig.76 Menú Interés

Al escoger la opción “Reportes” del menú “Interés” se muestra un listado de todos los intereses que se han determinado para realizar los préstamos.

INTERES			
Fecha Interes	Interes	Activa	
2011-12-03	0.18	S	Editar
2011-12-04	0.18	N	Editar
2011-12-04	0.18	S	Editar
2011-12-04	0.2	N	Editar
2011-12-04	0.17	N	Editar
2011-12-04	0.18	N	Editar
2011-12-01	0.19	N	Editar
2011-12-02	0.19	S	Editar
2009-01-01	0.12	N	Editar
2010-01-02	0.15	N	Editar

Fig.77 Reportes de Interés.

Además en cada interés existe un enlace “Editar” al hacer clic ahí se muestra en la ventana los datos del Interés seleccionado, que valores como la fecha, el valor de interés y si está activo o no, todos estos valores pueden cambiar. Una vez de acuerdo hacer clic en el botón “Guardar”.

Fig.78 Edición de datos de Interés

Al escoger la opción “Nuevo Interés” del menú “Interés, permite ingresar un nuevo valor de interés para los préstamos a realizar.

Debe ingresar el valor del Interés, la Fecha de activación de interés, y si está activo o no.

Una vez de acuerdo con los datos debe hacer clic en el botón “Guardar”.

Fig.79 Ingreso de datos de Interés

Tipo de cuenta

El menú “Tipo de Cuenta” tiene dos opciones:

- Reporte
- Nuevo Tipo de Cuenta.



Fig.80 Menú Tipo de Cuenta.

Al escoger la opción “Reporte” del menú “Tipo de Cuenta”, se muestra un listado de los Tipos de Cuenta para las agrupaciones de Estados de Cuenta, en este caso los Tipos de Cuenta son:

- Activo
- Pasivo
- Patrimonio
- Gasto
- Ingreso

TIPO DE CUENTA CONTABLE

Id Tipo Cuenta	Descripcion Tipo	
1	ACTIVO	Editar
2	PASIVO	Editar
3	PATRIMONIO	Editar
4	GASTO	Editar
5	INGRESO	Editar

Fig.81 Lista de Tipo de Cuenta Contable.

En cada Tipo de cuenta existe un enlace “Editar” al hacer clic se muestran los datos correspondientes al Tipo de Cuenta seleccionado, en la cual nos da la posibilidad de modificar el valor de la Descripción, una vez de acuerdo con los datos ingresados debe hacer clic en el botón “Guardar”.

EDICION DEL TIPO DE CUENTA CONTABLE

Id Cuenta Contable

Descripcion

Fig.82 Edición de Tipo de Cuenta Contable.

Al seleccionar la opción “Nuevo Tipo de Cuenta” del menú “Tipo de Cuenta”, permite ingresar los datos correspondientes a un nuevo Tipo de Cuenta en este caso un Identificador y la Descripción, una vez de acuerdo con los datos ingresados debe hacer clic en el botón “Guardar”.

Ingrese los Datos del Tipo de Cuenta

Id Tipo Cuenta

Descripcion

Fig.83 Ingreso de Datos de Tipo de Cuenta.

Parámetros

El menú “Parámetros” tiene las opciones:

- Parámetros de Aportes
- Monto de Préstamo

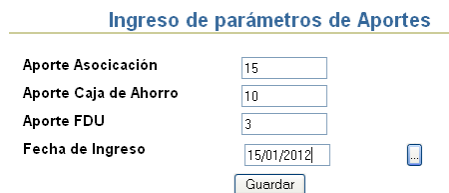


Fig.84 Menú Parámetros.

Al escoger la opción “Parámetros de Aportes” del menú “Parámetros”, permite ingresar los datos correspondientes a los aportes de:

- Aporte Asociación.
- Aporte Caja de Ahorro
- Aporte FDU

Además debe ingresar la Fecha de vigencia de los aportes.



Ingreso de parámetros de Aportes


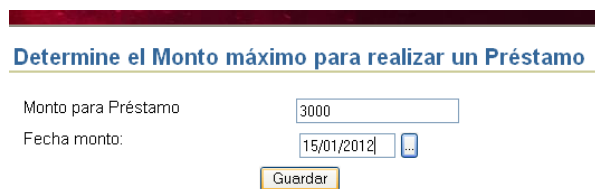
Aporte Asociación	<input type="text" value="15"/>
Aporte Caja de Ahorro	<input type="text" value="10"/>
Aporte FDU	<input type="text" value="3"/>
Fecha de Ingreso	<input type="text" value="15/01/2012"/> 

Fig.85 Ingreso de parámetros de Aportes.

Al escoger la opción “Monto de Préstamo” del menú “Parámetros”, permite ingresar el Valor máximo para realizar un préstamo a los socios y la Fecha en la cual se ingresa este valor, una vez de acuerdo debe hacer clic en el botón “Guardar”.



Determine el Monto máximo para realizar un Préstamo


Monto para Préstamo	<input type="text" value="3000"/>
Fecha monto:	<input type="text" value="15/01/2012"/> 

Fig.86 Ingreso de valor para el Monto máximo de un préstamo.

Usuarios

El menú “Usuarios” tiene las siguientes opciones:

- Socios
- Nuevo socio
- Nuevo Administrador



Fig.87 Menú Usuarios

Al escoger la opción “Socios” del menú “Usuarios” se muestra un listado de todos los socios Registrados.

LISTADO DE SOCIOS			
Id Socio	Nombres	Apellidos	Direccion
1000876472	JAIIME	AGUAS	Editar
0701033119	WIDMAR	AGUILAR GONZALEZ	Editar
1001354701	JAIIME	ALVARADO SANCHEZ	Editar
1000872109	HUMBERTO	BRAVO VALENCIA	Editar
1000993756	HECTOR RENE	BROWN	Editar
1102451887	JORGE	CARAGUAY PROCEL	Editar
1000074784	OCTAVIO	CEVALLOS VASQUEZ	Editar
1001133063	MARIA	DE LA PORTILLA VERA	Editar
1000785188	FERNANDO	EACHEVERRIA	Editar
1001584570	DARWIN JOSE	ESPARZA ENCALADA	Editar
1707852081	JOSE FERNANDO	GARRIDO SANCHEZ	Editar
1001139458	MILTON	GAVILANEZ VILLALOBOS	Editar
1001701180	PEDRO	GRANDA GUDIÑO	Editar
1000876421	JOSE	HUACA PINCHAO	Editar
1000811552	HUGO	IMBAQUINGO NARVAEZ	Editar
1001545142	EDGAR	JARAMILLO VINUEZA	Editar
1001303882	MARCELO	JURADO AVILA	Editar
1000000000	ENDERSON	LA RA REYES	Editar

Fig.88 Listado de Socios

En el nombre de cada socio existe un enlace llamado “Editar” al hacer clic sobre él se muestran los datos del Socio seleccionado, teniendo la posibilidad de modificar los Datos como: Nombres, Apellidos y Dirección.

Editar Datos del Socio	
id Socio	1000876472
Nombres	<input type="text" value="JAIIME"/>
Apellidos	<input type="text" value="AGUAS"/>
Direccion	<input type="text" value="av. el retorno"/>
<input type="button" value="Guardar"/>	

Fig.89 Edición de Datos de Socio

Una vez modificado los datos del Socio debe hacer clic sobre el botón “Guardar” y al revisar nuevamente el listado de los socios se pueden ver los cambios.

LISTADO DE SOCIOS			
Id Socio	Nombres	Apellidos	Direccion
1000876472	JAIIME	AGUAS	av. el retorno Editar
0701033119	WIDMAR	AGUILAR GONZALEZ	Editar
1001354701	JAIIME	ALVARADO SANCHEZ	Editar
1000872109	HUMBERTO	BRAVO VALENCIA	Editar
1000993756	HECTOR RENE	BROWN	Editar

Fig.90 Datos del socio modificado.

Al escoger la opción “Nuevo Socio” del menú “Usuarios”, da la posibilidad de ingresar los datos:

- Id de Socio - en este caso será el número de cédula.
- Password – Una contraseña para ingresar al sistema.
- Nombres – Los nombres del Socio.
- Apellidos – Los apellidos del Socio.

- Dirección – La dirección en la cual vive el Socio.

Una vez de acuerdo con los datos ingresados debe ingresar el botón “Guardar”.



Fig.91 Ingreso de datos del Socio

Acceso al sistema como Auditor

En la página de autenticación al ingresar el número de la cédula y la contraseña si es un usuario de tipo Auditor tiene las siguientes opciones: Parámetros y Datos. Además se muestra una página de bienvenida.



Fig.92 Página de Bienvenida y las opciones para un usuario de tipo Auditor.

Parámetros

El menú “Parámetros” tiene las opciones:

- Monto.
- Aportes.



Fig.93 Menú Parámetros.

Al escoger la opción “Monto” del menú “Parámetros” se va a desplegar un listado de la transacción que se ha realizado con el monto máximo de préstamo.

Se muestra el id de Monto, el usuario que realizó la transacción, el monto al momento, el monto que tenía antes, Fecha en la cual se realizó la transacción y el estado.

Transacciones de Monto Máximo de Préstamo						
IdMonto	Usuario	Transaccion	Monto Prestamo Actual	Monto Prestamo Anterior	Fecha	Estado

Fig.94 Transacciones de monto máximo de préstamo.

Al hacer clic en la opción “Aportes” del menú “Parámetros”, se muestran todas las transacciones realizadas en los Aportes que los socios realizarán.

Se muestra el Id Parámetros Aporte, el usuario quien realiza la transacción, la transacción que realizó (Ingreso, Edición), Aporte de Asociación actual y anterior, Aporte a la Caja actual y anterior, Aporte al Fdu actual y anterior, la fecha que realiza la transacción y el Estado de los aportes.

Transaccion de Aportes de Socios									
Id Parametro Aporte	Usuario	Transaccion	Aporte Aso Actual	Aporte Aso Anterior	Aporte Caja Actual	Aporte Caja Anterior	Aporte Fdu Actual	Aporte Fdu Anterior	Fecha Estado

Fig.95 Transacciones de los aportes de los Socios.

Datos

El menú “Datos” tiene las opciones:

- Interés.
- Periodo Contable.



Fig.96 Menú Datos.

Al escoger “Interés” del menú “Datos”, se muestra un listado de todas las transacciones de interés que se han realizado.

Se muestra el Id de Interés, el usuario, el tipo de transacción (Ingreso, Edición), Interés Actual y Anterior, Estado y Fecha de la transacción.

Transacciones de Interés						
Id Interes	Usuario	Transaccion	Interes Actual	Interes Anterior	Estado	Fecha
33	0401308069	Edicion	0.19	0.19	N	2011-12-01 00:00:00.0
24	0401308069	Ingreso	0.17	0.0	S	2011-12-04 00:00:00.0
27	0401308069	Ingreso	0.18	0.0	S	2011-12-04 00:00:00.0
24	0401308069	Edicion	0.2	0.2	N	
30	0401308069	Ingreso	0.19	0.0	S	2011-12-01 00:00:00.0
33	0401308069	Ingreso	0.19	0.0	S	2011-12-02 00:00:00.0

Fig.97 Transacciones de Interés.

Al escoger la opción “Periodo Contable” del menú “Datos”, se despliega un listado con todas las transacciones de Periodos Contables realizadas.

Se muestra el Id de Periodo, Usuario quien realiza la transacción, transacción (Ingreso, Edición), Fecha inicial y final y estado.

Transacciones de Periodos					
Id Periodo	Usuario	Transaccion	Fecha Inicial	Fecha Fin	Estado
4	0401308069	Edicion	2010-01-01 00:00:00.0	2010-12-31 00:00:00.0	0
6	0401308069	Ingreso	2012-01-02 00:00:00.0	2012-12-31 00:00:00.0	1

Fig.98 Transacciones de Periodos Contables.

Acceso al sistema como Socio

En la página de autenticación al ingresar el número de cédula y la contraseña si se trata de un usuario de tipo Socio, se muestra una página de bienvenida y el siguiente menú:

- Reportes.
- Actualización de Datos.
- Ganancia



Fig.99 Página de bienvenida del socio.

Reportes

El menú “Reportes” tiene las siguientes opciones:

- Balance de cuentas.
- Pérdidas y Ganancias.
- Estado de Situación Final.



Fig.100 Menú Reportes.

Al escoger la opción “Balance de Cuentas” del menú “Reportes”, se muestra un listado de todas las Cuentas Contables con los valores: Saldo Debe, Saldo Haber, Total Debe y Total Haber.

BALANCE DE CUENTAS

Nombre	Saldo Debe	Saldo Haber	Total Debe	Total Haber
DISPONIBLE	0.0	0.0	0.0	0.0
CAJA	0.0	0.0	0.0	0.0
CAJA GENERAL	11236.28	6822.65	4413.63	0.0
CAJA DE AHORROS	72116.7	64752.84	7363.86	0.0
EXIGIBLE	0.0	0.0	0.0	0.0
CUENTAS POR COBRAR A SOCIOS	0.0	0.0	0.0	0.0
CUENTAS POR COBRAR DRA. PAREDES	2613.5	2171.68	441.82	0.0
CUENTAS POR COBRAR ING. REASCOS	0.0	0.0	0.0	0.0
CUENTAS POR COBRAR ING. FUENTE	6554.44	4144.41	2410.03	0.0
CUENTAS POR COBRAR ING. ESPARZA	0.0	0.0	0.0	0.0
CUENTAS POR COBRAR LIC. ROMAN	2000.0	1353.88	646.12	0.0
CUENTAS POR COBRAR ING. ECHEVERRIA	0.0	0.0	0.0	0.0
CUENTAS POR COBRAR	0.0	0.0	0.0	0.0

Fig.101 Balance de cuentas – socios.

Al escoger la opción “Pérdidas y Ganancias” del menú “Reportes” se despliega el Reporte de las Pérdidas y Ganancias de la Caja General.

ESTADO DE PÉRDIDAS Y GANANCIAS CAJA GENERAL

INGRESOS	9950.58
+ Aportes Socios	8793.3
+Aportes FDU	1167.28
MENOS GASTOS	5422.65
- Gastos Varios	5422.65
= Utilidad	3360.6499999999996

Fig.102 Estado de Pérdidas y Ganancias Caja General.

Al escoger la opción “Estado de Situación Final” del menú “Reportes”, se muestra las Cuentas Contables con su respectivo valor agrupado por el tipo de Cuenta.

ESTADO DE SITUACION FINAL	
Activos	
DISPONIBLE	0.0
CAJA	0.0
CAJA GENERAL	4413.63
CAJA DE AHORROS	7363.86
EXIGIBLE	0.0
CUENTAS POR COBRAR A SOCIOS	0.0
CUENTAS POR COBRAR DRA. PAREDES	441.82
CUENTAS POR COBRAR ING. REASCOS	0.0
CUENTAS POR COBRAR ING. PUENTE	2410.03
CUENTAS POR COBRAR ING. ESPARZA	0.0
CUENTAS POR COBRAR LIC. ROMAN	846.12
CUENTAS POR COBRAR ING. ECHEVERRIA	0.0
CUENTAS POR COBRAR ING. HUACA	0.0
CUENTAS POR COBRAR ING. BRAVO	1648.61
CUENTAS POR COBRAR ING. ALVARADO	1238.88
CUENTAS POR COBRAR ING. LOPEZ	2336.26
CUENTAS POR COBRAR ING. JARAMILLO	0.01
CUENTAS POR COBRAR ING. ORQUERA	2408.95
CUENTAS POR COBRAR ECO. OVIEDO	2108.78

Fig.103 Estado de Situación Final –Socio

Al escoger la opción “Edición” del menú “Actualización de Datos”, se muestra los datos del Socio los cuales pueden modificarlos, una vez de acuerdo con los datos cambiados debe hacer clic en el botón “Guardar”.

Ganancia

El menú “Ganancia” tiene las siguientes opciones:

- Interés Ganado
- Estado de Cuenta



Fig.104 Menú Ganancia.

Al escoger la opción “Interés Ganado” del menú “Ganancia”, se muestran los valores de todos los aportes que ha realizado como: Aporte a la Caja, Aporte a la Asociación, Aporte al Fdu y la Ganancia que ha obtenido por su aporte a la Asociación.

APORTES	
El Interés ganado por sus aportes es:	267.35100202020203
Total de Aportes a la Caja de Ahorro:	790.0
Total de Aportes a la Asociación:	484.0
Total de Aportes al FDU:	94.0

Fig.105 Aportes del Socio y Ganancia.

Al escoger la opción “Estado de Cuenta” del menú “Ganancia”, se muestran todas las cuotas que adeuda, en el caso de que no tenga ningún préstamo no se muestran datos.

Cuotas Adeudadas

Interes	Principal	Saldo	Vencimiento
27.83	93.39	2133.32	2012-01-28
26.67	94.55	2038.77	2012-02-27
25.48	95.74	1943.03	2012-03-28
18.79	206.85	1296.56	2012-03-28
24.29	96.93	1846.1	2012-04-27
23.08	98.14	1747.86	2012-05-27
21.85	99.37	1648.59	2012-06-26
20.61	100.61	1547.88	2012-07-26
19.35	101.87	1446.11	2012-08-25
18.08	103.14	1342.97	2012-09-24
16.79	104.43	1238.54	2012-10-24
15.48	105.74	1132.8	2012-11-23
14.16	107.06	1025.74	2012-12-23
12.82	108.4	917.34	2013-01-22
11.47	109.75	807.59	2013-02-21
10.09	111.13	696.46	2013-03-23
8.71	112.51	583.95	2013-04-22
7.3	113.92	470.03	2013-05-22
5.88	115.34	354.69	2013-06-21
4.43	116.79	237.9	2013-07-21
2.97	118.25	119.65	2013-08-20
1.5	119.72	-0.07	2013-09-19

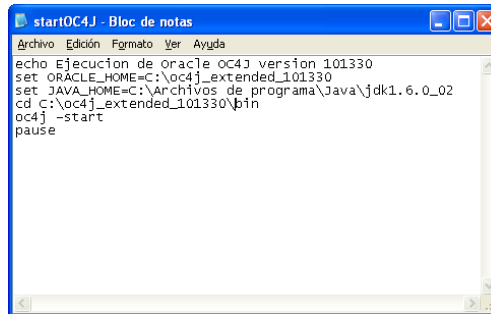
Fig.106 Cuotas Adeudadas.

MANUAL DE INSTALACIÓN

Oracle Application Server Standalone - OC4J

Para que el Servidor de Aplicaciones OC4J funcione debe realizar los siguientes pasos:

- 1.- Descomprimir el archivo oc4j_extended_101330.zip en la carpeta c:\oc4j_extended_101330.
- 2.- En la carpeta donde se descomprimió el archivo .zip, crear el archivo startoc4j.bat y editarlo



```
Archivo Edición Formato Ver Ayuda
echo Ejecución de Oracle OC4J version 101330
set ORACLE_HOME=C:\oc4j_extended_101330
set JAVA_HOME=C:\Archivos de programa\Java\jdk1.6.0_02
cd C:\oc4j_extended_101330\bin
oc4j -start
pause
```

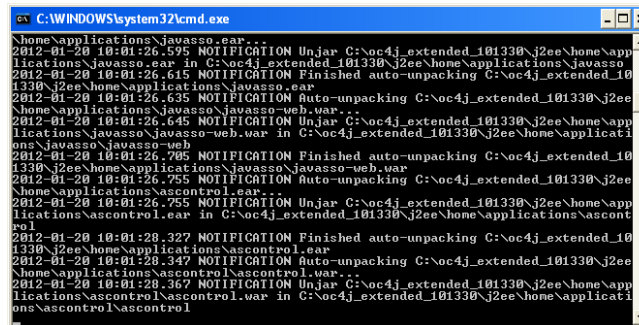
Fig.1 archivo startoc4j.bat

Ejecutar el archivo creado.



Fig.2 archivo startoc4j.bat - 2.

Al ejecutar el archivo solicita una contraseña para el usuario oc4jadmin; debe ingresarla, el usuario oc4jadmin es el administrador del servidor.



```
C:\WINDOWS\system32\cmd.exe
C:\home\applications\javasso.ear...
2012-01-20 10:01:26.395 NOTIFICATION Unjar C:\oc4j_extended_101330\j2ee\home\app
lications\javasso.ear in C:\oc4j_extended_101330\j2ee\home\applications\javasso
2012-01-20 10:01:26.615 NOTIFICATION Finished auto-unpacking C:\oc4j_extended_10
1330\j2ee\home\applications\javasso.ear
2012-01-20 10:01:26.635 NOTIFICATION Auto-unpacking C:\oc4j_extended_101330\j2ee
\home\applications\javasso\javasso-web.war...
2012-01-20 10:01:26.645 NOTIFICATION Unjar C:\oc4j_extended_101330\j2ee\home\app
lications\javasso\javasso-web.war in C:\oc4j_extended_101330\j2ee\home\applicati
ons\javasso\javasso-web
2012-01-20 10:01:26.705 NOTIFICATION Finished auto-unpacking C:\oc4j_extended_10
1330\j2ee\home\applications\javasso\javasso-web.war
2012-01-20 10:01:26.755 NOTIFICATION Auto-unpacking C:\oc4j_extended_101330\j2ee
\home\applications\ascontrol.ear...
2012-01-20 10:01:26.755 NOTIFICATION Unjar C:\oc4j_extended_101330\j2ee\home\app
lications\ascontrol.ear in C:\oc4j_extended_101330\j2ee\home\applications\ascont
rol
2012-01-20 10:01:28.327 NOTIFICATION Finished auto-unpacking C:\oc4j_extended_10
1330\j2ee\home\applications\ascontrol.ear
2012-01-20 10:01:28.347 NOTIFICATION Auto-unpacking C:\oc4j_extended_101330\j2ee
\home\applications\ascontrol\ascontrol.war...
2012-01-20 10:01:28.367 NOTIFICATION Unjar C:\oc4j_extended_101330\j2ee\home\app
lications\ascontrol\ascontrol.war in C:\oc4j_extended_101330\j2ee\home\applicati
ons\ascontrol\ascontrol
```

Fig.3 Ejecución archivo startoc4j.bat

Acceder a la página inicial del servidor escribiendo en el navegador:

<http://localhost:8888/>

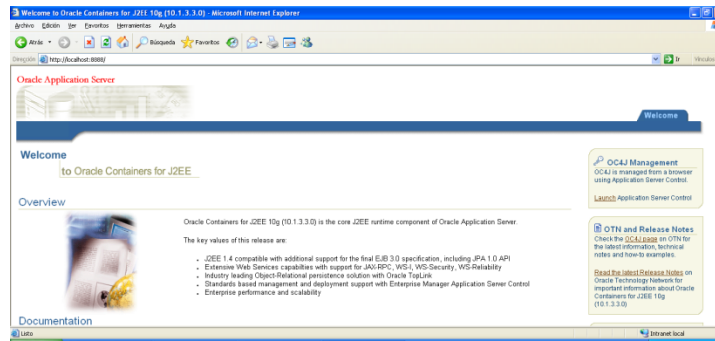


Fig.4 Acceso a navegador web

Luego debe hacer clic en el enlace “Launch”, se abre una nueva ventana en la cual debe ingresar: Nombre de Usuario y la contraseña.

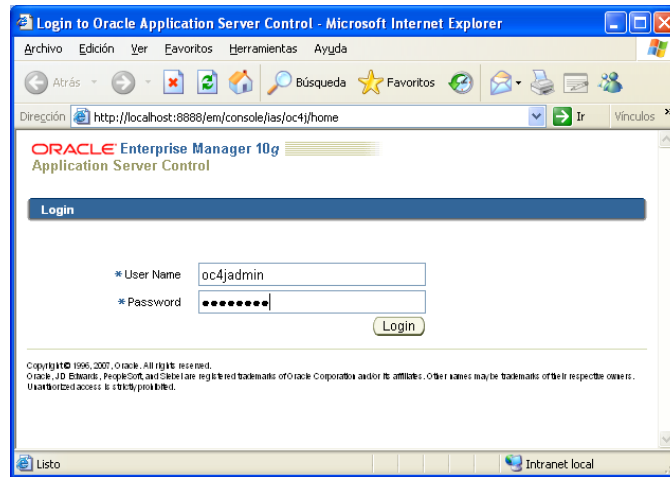


Fig.5 Autenticación de usuario oc4j

Accede a la configuración de oc4j, para subir la aplicación debe hacer clic en el enlace “Aplicaciones”.

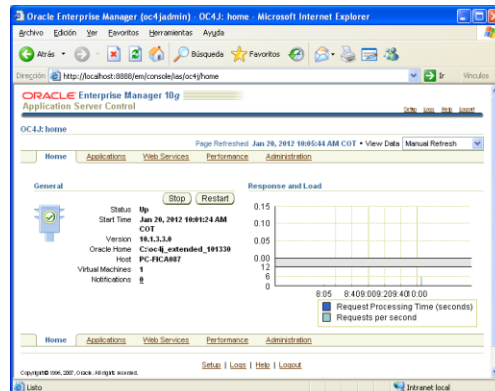


Fig.6 Administración oc4j

Para detener el servidor, CTRL+C en la consola de ejecución.

Instalación de Oracle 10g

Primero se debe configurar algunas opciones básicas de Oracle 10g sea en Windows XP, Windows 2000, Windows 2003, otros.

Descomprimir el archivo **Oracle RDBMS 10.2.0.1.zip** abrir la carpeta, ejecutar el fichero “setup.exe” para iniciar la instalación:

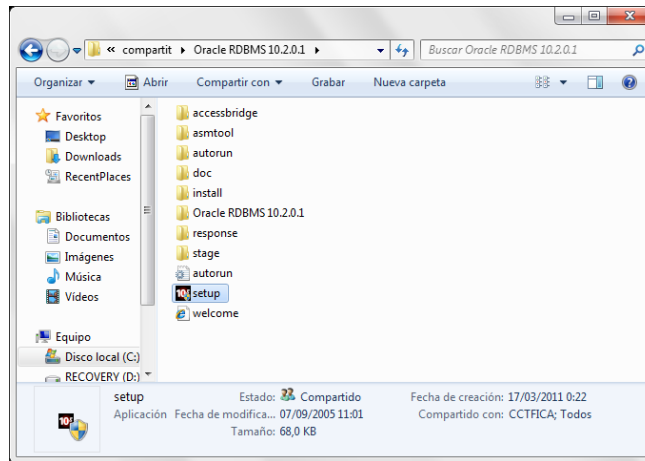


Fig.7 Archivos de instalación de Oracle

Seleccionar el método de instalación, en este caso “Instalación Básica”, y se deben especificar algunas opciones.

Ubicación del Directorio Raíz de Oracle: unidad y carpeta donde se realizará la instalación de Oracle 10g.

Tipo de instalación: EnterpriseEdition.

Crear base de datos inicial: creará una base de datos de uso general durante la instalación.

- **Nombre de la Base de Datos Global:** nombre con el que se identificará la base de datos. En este caso **FICA**.
- **Contraseña de Base de Datos:** contraseña que se asignará a los usuarios SYS, SYSTEM, SYSMAN y DBSNMP.

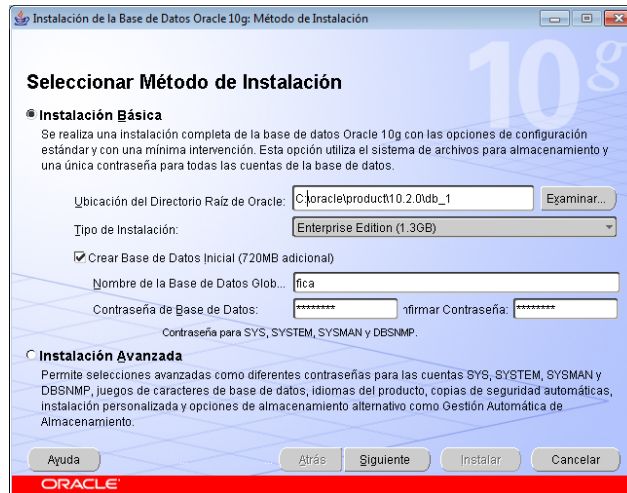


Fig.8 Instalación de Base de datos Oracle.

Luego de haber llenado estos datos pulsar "Siguiente" para continuar con la instalación. Se muestra el progreso de la instalación.



Fig.9 Proceso de Instalación

El asistente de instalación verificará si el entorno cumple todos los requisitos mínimos para instalar y configurar los productos seleccionados. Si hay algún elemento marcado con advertencia se deberá comprobar manualmente.

Pulsar "Siguiente" para continuar:



Fig.10 Comprobación de requisitos

Se muestra una ventana indicando los productos Oracle Database 10g 10.2.0.1.0 que se instalarán. Pulsar "Instalar" para iniciar la instalación:

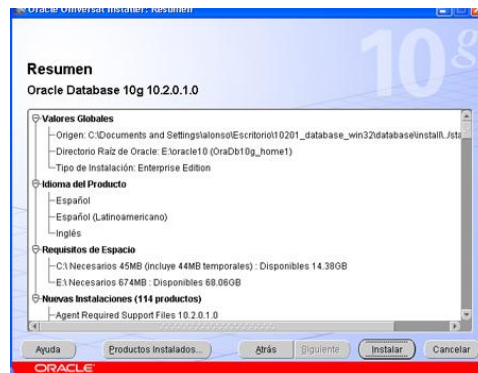


Fig.11 Resumen de Datos de Instalación

Se mostrará una ventana con el progreso de la instalación

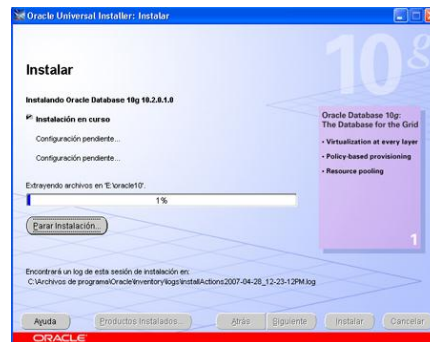


Fig.12 Progreso de la instalación.

Luego se mostrará el progreso de este proceso de la creación de la Base de Datos.

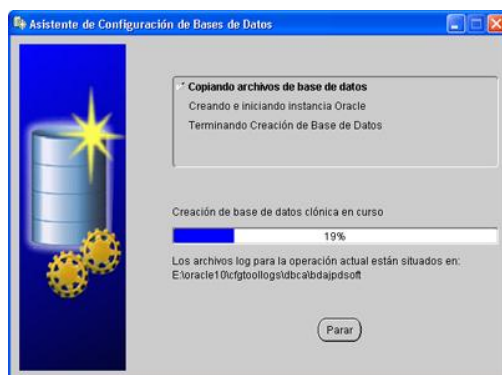


Fig.13 Creación de Base de Datos en curso.

Una vez creada la base de datos se mostrará una ventana para poder gestionar las contraseñas de cada usuario, también puede activarlos o desactivarlos, para ello pulsare en "Gestión de Contraseñas":



Fig.14 Gestión de Contraseñas.

Si se ha escogido la opción “Gestión de Contraseñas”, se muestra una ventana, donde puede activar/desactivar los usuarios que Oracle crea por defecto y cambiar sus contraseñas.

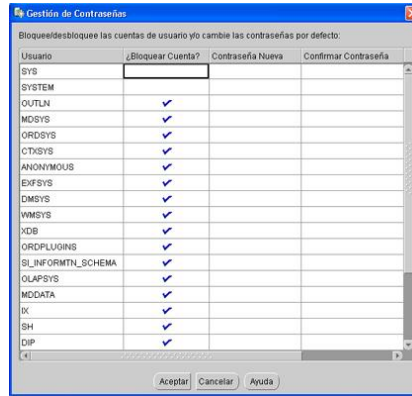


Fig.15 Selección de Cuentas Bloqueadas y gestión de contraseñas.

Hacer clic en Aceptar, regresa a la ventana inicial y nuevamente hacer clic en Aceptar.

Una vez este completa la instalación, el asistente mostrará una serie de datos. Pulsar en "Salir" para terminar la instalación.



Fig.16 Finalización de Instalación

Pulsare en "Sí" para cerrar el asistente de instalación de Oracle Database 10g:



Fig.17 Mensaje de salida de Instalación.

Para probar y configurar Oracle Database 10g, acceder al navegador de Internet e ingresar la siguiente dirección URL:

<http://localhost:1158/em>

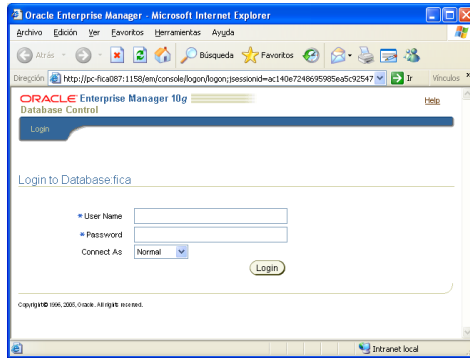


Fig.18 Accediendo a Oracle a través del navegador

Tras abrir esta URL del nuevo Oracle Enterprise Manager 10g, se mostrará una web en la que debe introducir el usuario y la contraseña para el acceso, en este caso utilizar el usuario "system". Pulsar el botón "Conectar":

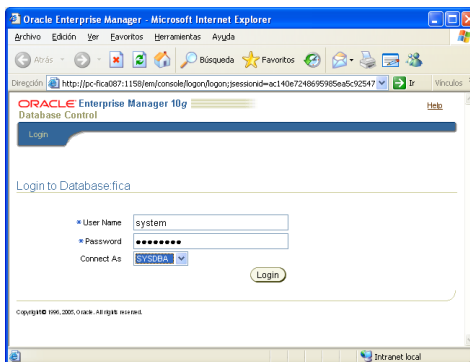


Fig.19 Conexión a la Base de Datos.

La primera vez que ejecute Database Control pedirá que lea y acepte el acuerdo de licencia. Si está de acuerdo con los términos de la licencia pulsar en "Acepto":



Fig.20 Acuerdo de Licencia.

El nuevo Oracle Enterprise Manager 10g Database Control mostrará una ventana inicial con una especie de cuadro de mandos, con las estadísticas de uso de la CPU, las sesiones activas, el estado de la base de datos, versión, host, listener, el nombre de la instancia, estadísticas sobre rendimiento de la base de datos, resumen de espacio, resumen de diagnósticos, etc:

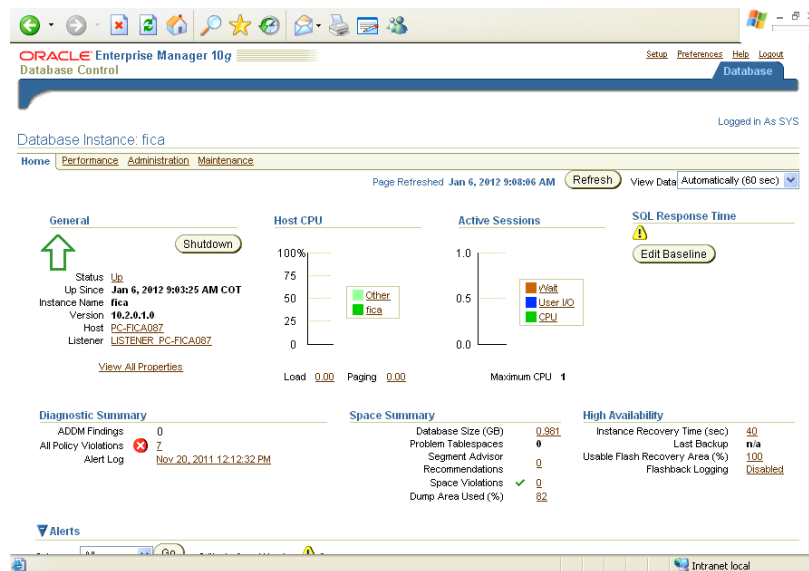


Fig.21 Resumen de la Base de Datos.

Se puede ver estadísticas de rendimiento en tiempo real pulsando en "Rendimiento":

Al pulsar en Tablespaces, se muestran los existentes. Por defecto, el instalador de Oracle 10g crea los siguientes tablespaces:

- EXAMPLE
- SYSAUX
- SYSTEM

- TEMP
- UNDOTBS1
- USERS

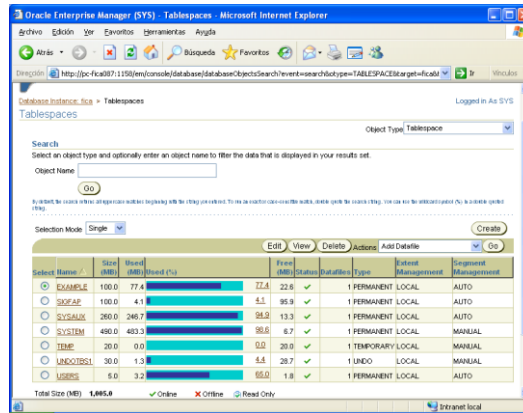


Fig.22 Tablespaces de la Base de Datos.

Debe crear un nuevo Tablespace para ello debe hacer clic en el botón “Create”, y en el nombre debe escribir “sigfap”, y especificar las características que desee que tenga.

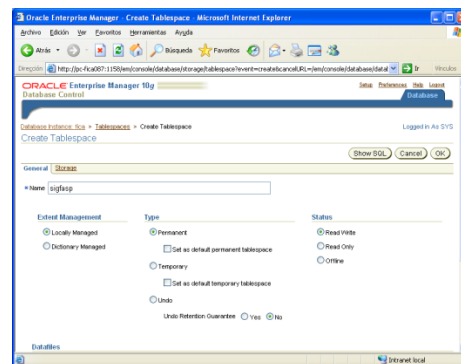


Fig.23 Creación del Tablespace sigfap

Luego debe especificar el usuario para la base de Datos, para ello en la Sección -> Administración->User&Privileges ->User

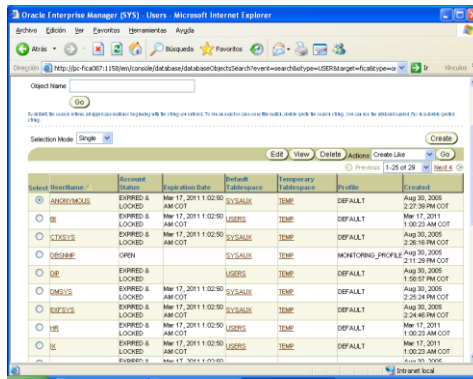


Fig.24 Usuarios de la Base de Datos

Hacer clic en el botón “Crear” e ingresar el nombre del usuario, ingresar la contraseña, el default tablespace y el temporarytablespace, luego hacer clic en el botón “OK”

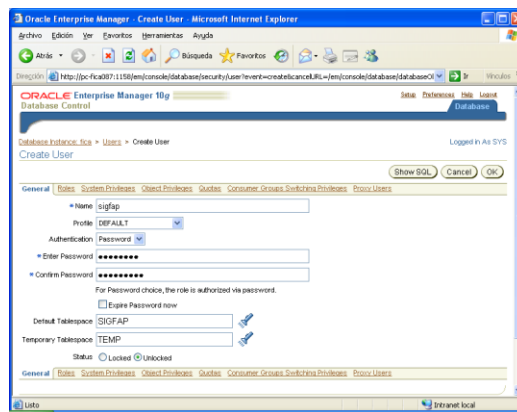


Fig.25 Usuariosigfap

En la sección "Mantenimiento" se puede: planificar copias de seguridad, realizar recuperaciones, gestionar copias de seguridad actuales, gestionar puntos de restauración, ver informes de copia de seguridad, valores de copia de seguridad, valores de recuperación, valores del catálogo de recuperación, exportar archivos de exportación, importar archivos de exportación, importar base de datos, cargar datos de archivos de usuario, clonar base de datos, etc.

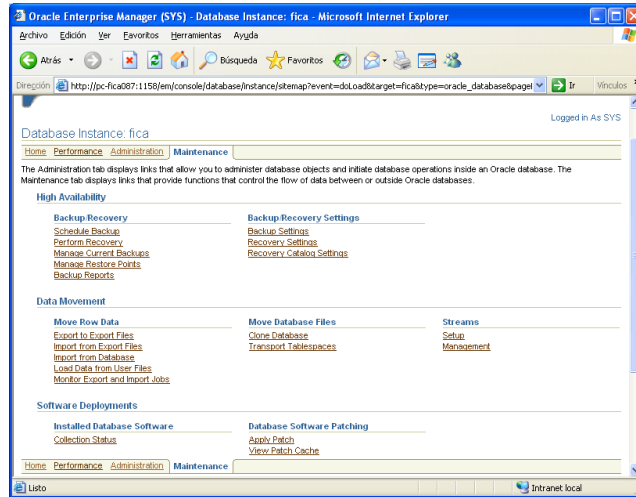


Fig.26 Opciones de Mantenimiento